

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**AN EXPLORATORY ANALYSIS OF WATERFRONT
FORCE PROTECTION MEASURES USING SIMULATION**

by

Matthew D. Childs

March 2002

Thesis Advisor:

Arnold H. Buss

Second Reader:

Matthew G. Boensel

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2002	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) An Exploratory Analysis of Water Front Force Protection Measures Using Simulation			5. FUNDING NUMBERS	
6. AUTHOR(S) Matthew D. Childs				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Since the <i>USS Cole</i> incident in October 2000, and particularly since the terrorist attacks of September 2001, Force Protection has become a fundamental issue. Of particular concern to the Navy is waterfront Force Protection: the protection of in-port High Value Units from attacks from the sea. The unpredictability of when or how a terrorist attack might be executed makes simulation an excellent tool for analyzing the waterfront force protection issue quantitatively. This thesis develops and implements a simulation model of Patrol Boats at the Naval Submarine Base in Bangor, Washington using Java and Simkit, both of which are platform independent, and therefore universally usable. The simulation is run pitting eight different notional Patrol Boat configurations (varying the number of patrol boats used, their intercepting and patrolling speeds, and their patrolling patterns) against eight notional terrorist attacks. The results of the simulation runs are analyzed, and general conclusions are drawn. The results indicate that the number of patrol boats used in an area and the speed they use to intercept threats are the most important factors of the four analyzed. Patrolling speed and patrolling patterns are found to be insignificant.				
14. SUBJECT TERMS Force Protection, Simulation, Simkit, Modeling			15. NUMBER OF PAGES 115	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**AN EXPLORATORY ANALYSIS OF WATER FRONT FORCE PROTECTION
MEASURES USING SIMULATION**

Matthew D. Childs
Lieutenant Commander, United States Naval Reserve
B.S., United States Naval Academy, 1989
M.A., University of Maryland (College Park), 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

**NAVAL POSTGRADUATE SCHOOL
MARCH 2002**

Author: Matthew D. Childs

Approved by: Arnold Buss, Thesis Advisor

Matthew Boensel, Second Reader

James Eagle, Chairman
Operations Research Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Since the *USS Cole* incident in October 2000, and particularly since the terrorist attacks of September 2001, Force Protection has become a fundamental issue. Of particular concern to the Navy is waterfront Force Protection: the protection of in-port High Value Units from attacks from the sea. The unpredictability of when or how a terrorist attack might be executed makes simulation an excellent tool for analyzing the waterfront force protection issue quantitatively. This thesis develops and implements a simulation model of Patrol Boats at the Naval Submarine Base in Bangor, Washington using Java and Simkit, both of which are platform independent, and therefore universally usable. The simulation is run pitting eight different notional Patrol Boat configurations (varying the number of patrol boats used, their intercepting and patrolling speeds, and their patrolling patterns) against eight notional terrorist attacks. The results of the simulation runs are analyzed, and general conclusions are drawn. The results indicate that the number of patrol boats used in an area and the speed they use to intercept threats are the most important factors of the four analyzed. Patrolling speed and patrolling patterns are found to be insignificant.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	AREA OF RESEARCH	1
B.	CONTEMPORARY NAVAL FORCE PROTECTION ISSUES	2
C.	OBJECTIVES AND RESEARCH QUESTIONS	3
D.	SCOPE OF THESIS AND METHODOLOGY	4
II.	SERVER AND MOVER EVENT BASED SIMULATION	7
A.	OBJECTS, DISCRETE EVENT SIMULATION AND EVENT GRAPHS	7
B.	COMPONENT BASED PROGRAMMING AND SIMKIT	13
C.	MULTIPLE SERVER QUEUES AND MOVERS	16
III.	MODEL DEVELOPMENT	23
A.	MODEL COMPONENTS	23
1.	Mover Managers	24
2.	Contact Generators	27
3.	TerroristMoverManager.....	29
4.	PBMM.....	30
5.	PBController.....	33
6.	BreachSensor	36
B.	OTHER CONSIDERATIONS.....	37
1.	Face Verification and Validation: Assumptions	37
2.	Graphics	40
IV.	MODEL RUNS AND ANALYSIS.....	43
A.	EXPERIMENTAL DESIGN.....	43
1.	Fundamentals: Repeatability and Replication.....	43
2.	Statistics	46
a.	<i>Number Intercepted</i>	46
b.	<i>Range at Intercept</i>	46
c.	<i>Number of Available Patrol Boats</i>	46
d.	<i>Number of Leakers</i>	47
e.	<i>Number of Terrors</i>	47
f.	<i>Number of Breaches</i>	47
g.	<i>Average Number in Queue</i>	47
3.	Run Setup: Fractional Factorial Design	47
B.	RESULTS AND ANALYSIS	53
1.	"Terrors" Regression Models.....	55
2.	"Leakers" Regression Models	61
3.	"Breaches" Regression Models	62
4.	"Range at Intercept" Regression Models	64
V.	CONCLUSIONS AND RECOMMENDATIONS.....	67
A.	ANALYTICAL CONCLUSIONS	67
B.	RECOMMENDATIONS.....	68

C.	FOLLOW-ON WORK	68
APPENDIX A.	JAVA CODE FOR THREAT INTERCEPT POINT	71
APPENDIX B.	DESIGN POINT DATA	75
APPENDIX C	LINEAR REGRESSION	83
APPENDIX D.	OUTPUT GRAPHICS	85
	LIST OF REFERENCES	95
	INITIAL DISTRIBUTION LIST	97

LIST OF FIGURES

Figure 1.	Force Protection Posture Sub Base Bangor Waterfront. After [Ref. 11]	5
Figure 2.	Fundamental Event Graph Construct	10
Figure 3.	Event Graph of the Arrival Process with Run	11
Figure 4.	Event Graph of the Multiple Server Queue	12
Figure 5.	Major Simulation Components and Listening Pattern.....	23
Figure 6.	PathMoverManager Event Graph (major elements)	25
Figure 7.	LoitererGenerator Event Graph (major elements)	28
Figure 8.	TerroristMoverManager Event Graph (major elements)	30
Figure 9.	PBMM Event Graph (major elements)	32
Figure 10.	PBController Event Graph (major elements).....	34
Figure 11.	BreachSensor event graph.....	37
Figure 12.	Graphical Display of Simulation with Notation Added.....	41
Figure 13.	Patrol Patterns: 4 PB's with bow-tie pattern and 2 PB's with barrier pattern shown	50
Figure 14.	Attack by One Dumb Terrorist	50
Figure 15.	Synchronized Attack by Two Smart Terrorists	51
Figure 16.	Coordinated Attack by One Smart and One Dumb Terrorist	51
Figure 17.	Graphical Confirmation of Mean Regression Result	57
Figure 18.	Graphic Showing the Advantage of Four Patrol Boats Over Two	57
Figure 19.	Graphic Showing the Advantage of Higher Intercept Speed	58
Figure 20.	Changes in Patrolling Speed Have No Effect Upon Output	58
Figure 21.	Changes in the Patrol Pattern Have No Effect Upon Output	59
Figure 22.	No Interaction Between Significant Factors	59
Figure 23.	Relative Effects of Significant Factors Upon P(terror) Outcome	60
Figure 24.	Design Point Six Always Best	85
Figure 25.	Advantage of Four Patrol Boats Over Two	85
Figure 26.	Advantage of Higher Intercept Speed	86
Figure 27.	No Clear Difference In Output	86
Figure 28.	No Clear (Significant) Difference in Output	87
Figure 29.	No Interaction Between Significant Factors	87
Figure 30.	Design Points Six and Eight Are Best	88
Figure 31.	Advantage of Four Patrol Boats Over Two	88
Figure 32.	Advantage of Higher Intercept Speed	89
Figure 33.	No Clear Difference in Output.....	89
Figure 34.	No Clear Difference in Output.....	90
Figure 35.	No Interaction Between Significant Factors	90
Figure 36.	Design Points Six and Eight Are Best	91
Figure 37.	Advantage of Four Patrol Boats Over Two	91
Figure 38.	Advantage of Higher Intercept Speed	92
Figure 39.	No Clear (Significant) Difference in Output	92

Figure 40.	No Clear Difference in Output.....	93
Figure 41.	No Interaction Between Significant Factors	93

LIST OF TABLES

Table 1.	Representation of Repeatability and Replication within an Experiment	44
Table 2.	Fractional Factorial Design of Complete Experiment	52
Table 3.	Output Data for Design Point One	53
Table 4.	Data for Regression Model for "Terrors" Statistic	55
Table 5.	Regression Output for "Terrors" Mean.....	56
Table 6.	Regression Output for "Terrors" Total Variance	61
Table 7.	Data for Regression Model for "Leakers" Statistic	61
Table 8.	Regression Output for "Leakers" Mean.....	61
Table 9.	Regression Output for "Leakers" Total Variation	62
Table 10.	Data for Regression Models for "Breaches" Statistic	62
Table 11.	Regression Output for "Breaches" Mean.....	63
Table 12.	Regression Output for "Breaches" Total Variance	63
Table 13.	Data for Regression Models for "Range at Intercept" Statistic	64
Table 14.	Regression Output for "Range at Intercept" Mean.....	64
Table 15.	Regression Output for "Range at Intercept" Total Variance	65
Table 16.	Design Point One Data.....	75
Table 17.	Design Point Two Data	76
Table 18.	Design Point Three Data	77
Table 19.	Design Point Four Data.....	78
Table 20.	Design Point Five Data	79
Table 21.	Design Point Six Data	80
Table 22.	Design Point Seven Data	81
Table 23.	Design Point Eight Data.....	82

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank CDR Matthew Boensel for his careful reading and editing, and especially for his brutal honesty when I least wanted to hear it.

I would like express my gratitude to Professor Arnold Buss for his hours of coding assistance into the wee hours of the morning, helping me to transform the shadows of an idea into a reality—albeit a virtual one.

I would like to thank my wife, Kelly, and my children, Eliot, Austin, Isabella, and Helena—born in the midst of it all—for their patience with Daddy’s absences.

I would most of all and always wish to express my deepest gratitude to Our Lady, the Seat of Wisdom, for her intercession on my behalf in helping me through the past two years intact: *Figlia del tuo figlio, Queen of Heaven.*

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The terrorist attacks of 11 September 2001 have made it clear that Force Protection must be a primary concern for our military force commanders. The threat of attacks on US soil is now a reality. Just as real is the threat of attacks on US waterways, and specifically upon in port high value US Naval units. Ships that are very potent weapons upon the high seas are very vulnerable targets when tied to piers. Analysis of the use of waterborne force protection assets--specifically patrol boats--can provide valuable insights into how high value Naval units can be effectively protected from attacks from the sea.

This thesis undertakes such an analysis using simulation. The specific setting of its analysis is Naval Submarine Base, Bangor, Washington, but the simulation can be applied to any waterfront area to allow for analysis of patrol craft operations. Simulation is an analysis tool especially suited to the terrorist attack problem, since terrorist attacks follow no particular (deterministic) patterns. Simulation, in this setting, allows for data collection in the absence of real data, by generating scenarios of interest, and seeing how patrol boats fare. Insofar as parameters--such as patrol boat speed, traffic intensity, and terrorist attack tactics--used within a simulation model are accurate reflections of reality, the results of the model provide real insights into force protection effectiveness.

The simulation within this thesis uses state-of-the-art techniques and software to model the waterfront at Sub Base Bangor. The basic waterfront is modeled, and is populated with patrol boats and randomly arriving and maneuvering vessels (i.e. traffic). Patrol boats remain in their patrolling patterns at low patrolling speeds, unless a contact moves inside the waterfront's exclusion zone, at which time a patrol boat accelerates to intercept it. After a time delay, a terrorist attack is attempted by either one or two attack craft--which, of course, are not known to be attack craft. Within the model, patrol boats are deployed in eight different configurations (varying their numbers, patterns and speeds) to protect the base's "delta" pier at which Trident submarines are moored. Each of the different patrol boat configurations is run a number of times against eight different kinds of notional terrorist attacks, differing in the number of attackers and their

approaches. In this way, the patrolling tactics can be compared across the same range of challenges so more successful configurations can be identified.

Analysis of the simulation output provided useful insights. Specifically, the model runs indicated that the keys to successful defense of the pier against a range of terrorist attacks are increasing the number of patrol boats deployed--to the maximum of four analyzed within the model--and increasing the speed at which the patrol boats intercept targets identified as threats. It may seem a computer simulation is not necessary to produce that insight. However, analysis of the simulation also indicated that neither patrolling speed (analyzed at two and five knots) nor the patrolling pattern (analyzed as a barrier and a bow tie or race track) of the boats had a significant effect upon outcomes. This implies that simple patterns and low patrolling speeds can be used. Both results have practical import for operators and equipment. A simple pattern allows for more concentration upon contact monitoring, and slower speeds may well result in greater fuel efficiency and fewer maintenance needs.

Analysis of the notional data was accomplished both through linear regression and, somewhat more intuitively, through graphical displays using a spreadsheet. Enhancements to the simulation could allow decision makers to run the model directly, defining simulation parameters. The exploratory analysis of the thesis can be used as a guide for analyses of patrol boat configurations using real data. Hence, in addition to the insights gained through the aforementioned analysis, the thesis provides both a tool and a methodology for analyzing the waterfront force protection problem.

I. INTRODUCTION

A. AREA OF RESEARCH

In recent times, attention has been being drawn to the threat of asymmetric and specifically terrorist attacks upon US military targets. Both *JV 2020* and the 1997 Report of the National Defense Panel, *Transforming Defense: National Security in the 21st Century*, cite asymmetric threats as a primary concern for strategy and force planning considerations in the twenty-first century. *JV 2020* posits that in the face of US conventional force dominance “the appeal of asymmetric approaches and the focus on the development of niche capabilities will increase”[Ref. 1; p. 4]. The National Defense Panel, in like manner, notes that would-be aggressors “may find new ways to attack our interests, our forces, and our citizens”[Ref. 2; p. 11]. Other literature concerned with the emerging nature of warfare—what we can now call “postmodern” warfare—is even more blunt and stark about what we must be able to defend and/or fight against in our times. Martin van Crevald, in discussing postmodern warfare writes: “War will not take place in the open field . . . Its normal *mise en scene* will be complex environments, either those provided by nature or else the even more complex ones created by men. It will be a war of listening devices and of car-bombs. . .”[Ref. 3; p. 626]. Carter et al., in their article *Catastrophic Terrorism*, raise the specter of large-scale terrorist attacks and call for counter-terrorist systems and postures that “highlight defensive needs before an incident happens”[Ref. 4; p. 83]. The terrorist attacks in New York, Washington, and Pennsylvania in September 2001 have certainly made the fact of the terrorist threat real, and their targeting of civilian centers adds universality to the struggle against terrorism, which is no longer to be confined to military arenas. They also made clear that we have not adopted the proper pro-active deterrent or defensive posture, at least at the national level. The war that those 11 September attacks began can only be waged effectively if proper force protection measures are established that can provide good defense in safe havens (e.g. ports) while offensive strikes are carried out elsewhere. Those measures need to be developed not by best guesses, but by specifically tailored quantitative analyses that can provide the best possible force protection measures for a given location

under specified operating conditions, assumed threat dispositions, and defined rules of engagement.

B. CONTEMPORARY NAVAL FORCE PROTECTION ISSUES

The 11 September attacks notwithstanding, we have known for quite some time, and very specifically since the USS *Cole* incident of October 2000, that high value and highly visible Naval targets are vulnerable to terrorism (particularly when in port), and that more robust force protection measures must be put in place. In his discussion of the *Cole* incident and recommendations made in its aftermath, retired Admiral Harold W. Gehman Jr. confirmed that van Crevald's description of the postmodern, asymmetric and terrorist age of warfare is accurate and will remain so: "The *Cole* was attacked by a truck bomb. It had an outboard motor, but it was a truck bomb . . . So what is the next attack going to look like? It is going to be a truck bomb in daylight, and it is going to be applied in some new and different way . . . Essentially, the bottom line of our force protection recommendations was to raise the cost of doing business in high threat areas"[Ref. 5]. Adm. Gehman's conclusion was disastrously borne out by the "truck bombs" that flew into the Trade Center towers and the Pentagon, so it stands to reason that our force protection efforts must be re-doubled to defend against that very sort of physical attack. David Weeks further focuses our attention upon the waterfront: "Since the devastation at Khobar Towers in 1996, the US. armed forces had dramatically increased their force protection and antiterrorism posture. Unfortunately for the Navy, the effort centered on the area from the main gate to ship quarterdecks, with very little attention paid to security from the waterside of the fleet—the gap, or 'seam,' referred to by former Secretary of Defense William Cohen in his introduction to the *Cole* Commission Report"[Ref. 6]. Even prior to 11 September, the Chief of Naval Operations (CNO) had directed Commanders to re-enforce that seam. His guidance clearly shows that force protection of vital naval assets is a primary and pervasive concern: "Force Protection considerations must become a primary planning factor for every mission, activity, or event—in-CONUS or out-of-CONUS. All commanders and commanding officers must fully integrate this key enabling factor into every operation." [Ref. 7]. Force Commanders have, in fact, begun to address the gap. In particular, Commander Submarine Forces Atlantic

(ComSubLant) and Commander Submarine Forces Pacific (ComSubPac) have focused their attention—particularly after the *Cole* incident—upon Trident Class or, more generally, Ballistic Missile Nuclear Submarine (SSBN) in-port Force Protection measures. To this end the Johns Hopkins Applied Physics Laboratory was contracted by the Chief of Naval Operations Submarine Division, Science and Technology Branch (N775), to develop the In-Port/Near-Port Project, which was launched to improve force protection of US ballistic missile submarines:

The In-Port/Near-Port Project is a new project focusing exclusively on in-port and near-port asymmetric vulnerabilities of the SSBN force. Four tasks—[1]vulnerability characterization, [2]vulnerability assessment, [3]countermeasures assessment and concept development, and [4]risk assessment, comprise this project...The Countermeasure Assessment and Concept Development Task will assess existing countermeasures and develop new operational countermeasure concepts. Recommendations may range from changes in operating procedures to new systems concepts...[Ref. 8].

The message traffic from Sub Base Bangor after the 11 September attacks makes it clear force protection analyses already underway have a new sense of relevance and urgency:

On September 11th the world changed. The terrorist attacks against the citizens of our country painfully revealed the vulnerabilities of freedom. As Commanding Officers, you need to be thinking about the vulnerabilities of your ship and crew. 2. The day to day readiness of the Trident Submarine Force to conduct its assigned mission is a critical element of our nation's security. Survivability has always been our top priority, but now, we must evaluate our practices for relevance. You trained you crew for the ASW [Anti-Submarine Warfare] threat, you now have the challenge to prepare them to deal with terrorism. Make no mistake about it, the terrorist threat is real. [Ref. 9]

C. OBJECTIVES AND RESEARCH QUESTIONS

Clearly, the overarching fleet force protection objective is effective protection of vital pier-side assets. The objective of this thesis is to contribute to that overall effort by quantifying the basic elements of waterfront force protection assets, possible waterborne attackers, and their interaction through a model/simulation. This effort is meant to augment and complement the Johns Hopkins effort. The Johns Hopkins database is a

useful descriptor of base layout and extant force protection assets and procedures, but it provides no calculation, recommendations or analyses to increase force protection effectiveness. It does nothing substantial to address the third (countermeasures assessment and concept development) and fourth (risk assessment) tasks of the In-Port/Near-Port Project. This thesis investigates steps that can be taken to strengthen waterfront force protection at Sub Base Bangor, WA, the homeport of Trident Submarines (SSBN's) under Commander Submarine Group Nine (CSG-9). To help CSG-9—and by extension any waterfront commands and even, more broadly, any specific location—more effectively and proactively protect assets, a method of analyzing different force protection configurations' effectiveness against a distribution of possible attacks must be developed.

By focusing upon specific geographies and feasible force distributions at specific waterfront locations, a limited number of possible scenarios can be subjected to quantitatively meaningful analysis. Even so, the terrorist attack problem presents a difficulty to the analyst. Because of the random and isolated nature of terrorist attacks, there simply are no data sets describing the effects of terrorist attacks, per se, on a given kind of target. All one can say is how harmful a given, unrepeated—and often unrepeatabe—attack was; one cannot generate statistics. The terrorism problem, in other words, cannot be analyzed using traditional means. The best hope one has in meaningfully dealing with the very real threat of waterfront terrorist attacks quantitatively is through simulation[Ref. 10; p. 91]. This thesis will develop such a simulation to allow investigative analysis of force protection asset configuration and thereby address the tasking of N775 to “assess existing countermeasures and develop new operational countermeasure concepts.”

D. SCOPE OF THESIS AND METHODOLOGY

The simulation will give insight into the potential success of defensive patrol boat units in response to different terrorist threats. It will, therefore, serve as an exploratory tool in the development of patrol and tactical approaches under geographical and equipment constraints and existing rules of engagement. The simulation will make it

possible to compare different tactical approaches under the given MOE of preventing any successful intercept of an SSBN by a Threat, and to determine if different tactics lead to statistically different outcomes. The simulation will not and cannot predict either the onset or outcome of an attack; but it can be used effectively to gain insight into robust force protection postures and resource needs by focusing on the waterborne element, and to demonstrate what kinds of analyses can be accomplished with its output.

The physical basis of the model is the waterfront geography at Naval Submarine Base Bangor, Washington, centered on the “Delta” pier at which SSBN’s are moored.

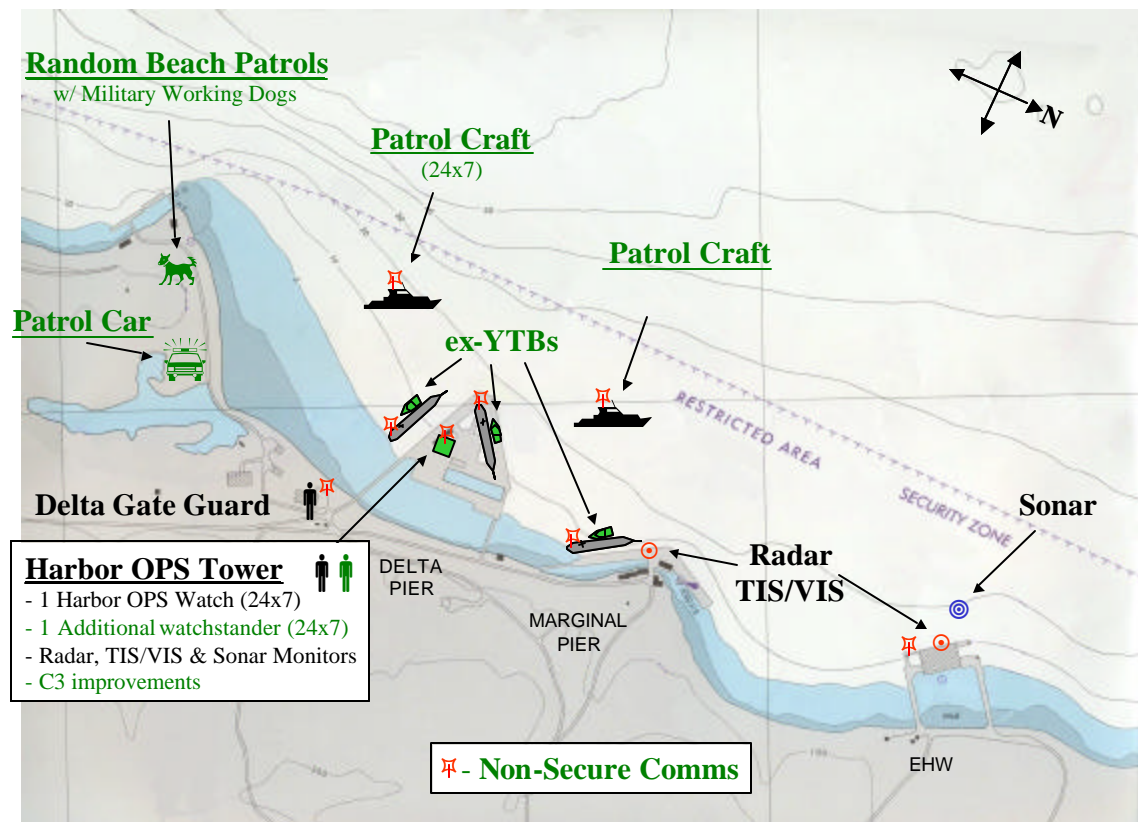


Figure 1. Force Protection Posture Sub Base Bangor Waterfront. After [Ref. 11]

The geography is modeled as a rectangle defined in its length (North-South axis) by sensor capacity and in its width (East-West axis) by the width of the river. CSG-9 initially had up to two contracted patrol boats on guard duty in the vicinity of Delta Pier. In the aftermath of 11 September, more assets have been committed. Exploratory analysis of the utilization of the boats is the primary focus of the simulation, though their

performance is closely tied to detection equipment, the location and performance of which can be varied with model expansion to a non-line of sight problem.

The model will be implemented in Java[Ref. 12] and Simkit[Ref. 13]. These are both freely available tools and can be executed on any operating system, thanks to Java's platform independence. Simkit is the result of earlier work done at the Naval Postgraduate School by K. Stork in his thesis, *Sensors in Object Oriented Programming with Java: An Introduction*. [Ref. 14]

II. SERVER AND MOVER EVENT BASED SIMULATION

A. OBJECTS, DISCRETE EVENT SIMULATION AND EVENT GRAPHS

The Force Protection simulation described herein is written in Java, an object-oriented language. Object-oriented programming languages stand in contrast to *imperative programming languages* or *structured programming languages*, like FORTRAN or BASIC. [Ref. 15; p. 15] The essential characteristic of object-oriented programming, and that which makes it a natural fit for simulation, is that it "considers a problem to consist of collections of *objects* and the various interactions between them. These objects often model artifacts that are present in the real world [Ref. 15; p. 35]. The objects within a program can assume a state and transition from one state to another based upon what happens in the simulation. Object-oriented programming is therefore ideal for simulation implementation because it has an intuitive correspondence with the reality being modeled. It is much more sensible, for example, to develop a simulation of Patrol Boats against Threats using objects called Patrol Boats and Threats, which behave in certain programmed ways, than to deal with mere variables or mathematical expressions. Beyond that, object-oriented programming allows for the re-use of given objects, or software components, because objects are self-contained entities that can either be used in different simulations as they stand, or be adapted or "extended," in Simkit terms, to fit a role only slightly different from a previous function. Because object-oriented programming models problems in terms of objects transitioning from one state to another, it lends itself to Discrete Event Simulation, which is really nothing more than what it says (simulation realized through the tracking of successive events) and is specifically the time –varying state of a simulation's component objects.

The following general discussion is based upon Law and Kelton [Ref. 10] and Buss [Ref. 16]. Discrete Event Simulation (DES), or modeling, is built upon two fundamental elements, "a set of state variables, and a set of events" [Ref. 16]. The state variables as a collective do what their name suggests: they define the state of the system being modeled at any given time. The choice of state variables in any DES is very important because those variables alone will be what represent the reality being modeled—the variables chosen, in other words, must be sufficient to capture the

important features or components of the simulated reality. The events in a DES are the changes the state variables undergo at specific (instantaneous) moments in time. Most events change the state of the system, while others do not—the event that stops a simulation, for example, has no effect on the state variables. A DES simulates the reality or system being modeled by generating “state trajectories,” or plots of state variable values over time. These state trajectories provide statistics for state variables that allow analysis of system—and, by extension, system component—performance. The way a system performs relies entirely upon when and how frequently various events occur. In fact, unless one wishes to simulate total inactivity, it is precisely the timing of successive events undertaken by key elements (read state variables) that defines the entire simulation. Hence, for the two components of DES actually to *be* a simulation, they must have some sort of engine, some scheduler by which a simulation moves from *potens* to *actus*. This engine is the Future Event List.

Buss puts the matter succinctly: “the Future Event List (or simply the Event List) . . . is nothing more than a ‘to-do’ list of scheduled events”[Ref. 16]. As with all to-do lists, it is constantly being changed or updated based upon current events—like the meeting that becomes necessary between your next two appointments as a result of the phone call you just received. The future event list is perhaps the singular feature—from whence it derives its name—of DES that distinguishes it from time-stepped simulation, in that it advances simulation time at varying intervals determined by scheduled events rather than advancing time in constant sized steps regardless of whether anything is happening in the simulation at the next regular time step or not. In DES, progress is driven by events, not time. The timing of events is certainly vitally important, since it affects the relative state trajectories of the systems’ components, and therefore the state of the system at large; but that very timing is a function of events that need to be accomplished. Because of the event list, a Discrete Event’s Simulation’s progress is actually very conceptually simple; it just moves from one event to the next event on the Event List until there are no events left—or until it is told to execute an event that tells it to stop. In other words, at any single moment, a DES is only concerned, so to speak, with the event being executed and the next scheduled event: its field of vision is one event

long—this is the very reason Event Graphs are so useful in describing DES, as discussed below. Law and Kelton summarize the Event List driven progress of a DES quite nicely:

The simulation clock is initialized to zero and the times of occurrence of future events are determined. The simulation clock is then advanced to the time of occurrence of the *most imminent* (first) of these future events, at which point the state of the system is updated to account for the fact that an event has occurred, and our knowledge of the times of occurrence of future events is also updated [note that the event is instantaneous]. Then the simulation clock is advanced to the time of the (new) most imminent event [which may have been scheduled upon execution of the last event and inserted before all others in the list], the state of the system is updated, and future event times are determined, etc. [Ref. 10]

This DES process is more efficiently captured by the discrete event algorithm, by which the future event list is managed. In pseudo-code the algorithm looks something like this:

```
While (Event List is not empty)
  Advance time to next event
  State transition for event
  Remove event from list
  Schedule other event(s) (if necessary)
```

Event graphs are the most efficient means of showing how an event-based simulation is constructed. They allow for graphical abstraction of event-based models, “a way of representing the Future Event List logic for a discrete-event model”[Ref. 18]. The event graph was first developed by L. Schruben in 1983 [Ref. 17] and is discussed in detail by A. Buss. Buss has specifically used the event graph construct in demonstrating some of the basic functionality of Simkit, which is the simulation program used in this thesis.

As with DES, event graphs have two essential elements, in this case nodes and directed edges: “Each node corresponds to an event, or state transition, and each edge corresponds to the scheduling of other events. Each edge can optionally have an associated Boolean condition and/or a time delay”[Ref. 17 and Ref. 18]. The construct

demonstrates the progression through the Event List as described above, by presenting the movement from one event to the next pictorially. The inclusion of conditions and time delays allows for a virtually complete demonstration of a program's logic without the necessity of having recourse to the code that implements it. The linking of a number of Event Graphs is an ideal way to show not only the flow within a given program or "class" of a simulation, but as well the linking and interaction of component classes of that program, which is vital in both the development and presentation of an object-oriented environment, and will be discussed further below. The most basic relationship of Events is demonstrated by "the fundamental event graph construct:"

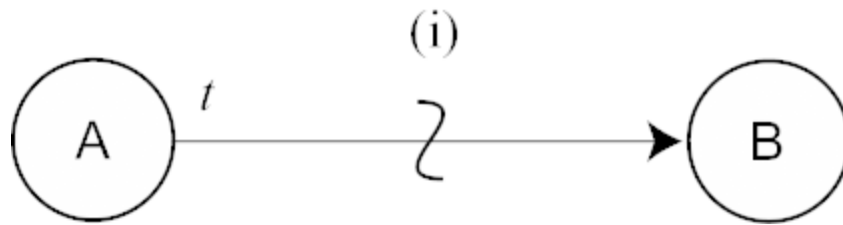


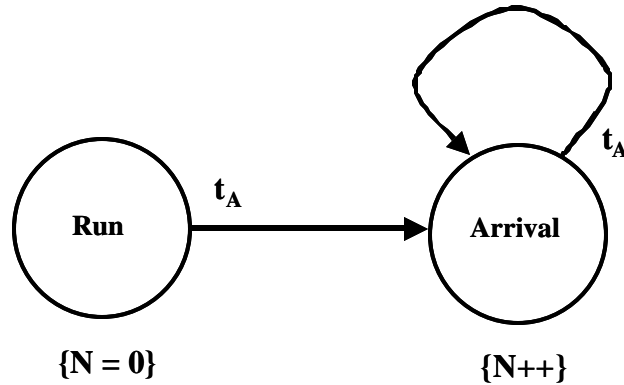
Figure 2. Fundamental Event Graph Construct

Nodes A and B represent events or methods. The small *t* shows there is a time delay, and the wavy line on the edge shows there is a Boolean condition, (i), that must be met for event B to occur. Hence, the construct as a whole says that the occurrence of Event A causes Event B to be scheduled for execution after a time delay of *t*, provided the logical condition (i) is met.

Once the fundamental event graph is understood, other event graphs are, relatively easily, recognized as extensions of that principle construct. Once again, the important point is that any event graph is simply an abstraction of the events of an Event Based Simulation, which in turn means it is simply an illustration of how and in what order Events are placed on the Future Events List. Anything that a modeler wants to have happen in a given simulation can happen only if it is scheduled, and it will only be scheduled if the programmer has provided a method to schedule it. Once again the power of the Event Graph is a clear help: the only way an event can occur is if there is a directed

path of scheduling edges from the Run event to it. Two more specific Event Graphs will provide sufficient background for specific application to the PatrolBoat model. Both are pictures of general processes or classes that are fundamental to the Force Protection simulation. The first is the “Arrival Process,” which, as the name suggests, simply “does” arrivals: it is a class that counts the number of arrivals of a given type of object onto the stage of a simulation, and schedules follow-on arrivals.

Figure 3. Event Graph of the Arrival Process with Run



In this case, the Arrival Process is shown with the “Run” method attached. The Run method is a vital part of any Simkit simulation program, because it is the method that indirectly resets state variables—as in this case, where “N”, the number of arrivals, is set to zero—and schedules the simulation’s first event. The Arrival Process Event Graph (with the Run method attached) is interpreted thusly: upon the execution of the Run method, the number of arrivals, N, is set to zero, and the first Arrival is scheduled for—placed on the Future Events List to occur at—time zero plus t_A . Since there is no logical condition to be met, the “do” Arrival method is executed, which means that at time t_A the value of the state variable N is incremented a single unit and the next Arrival is scheduled to occur at time zero, plus t_A , plus t_A . The process is simply repeated until the program is halted as discussed above.

The second basic embellishment of the Basic Event Graph Construct (which includes that of the Arrival Process) is the Event Graph of the Multiple Server Queue—the driving conceptual construct of the PatrolBoat simulation.

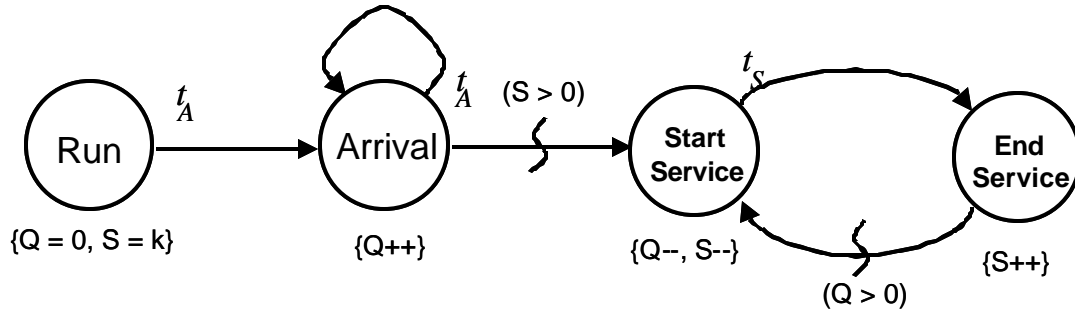


Figure 4. Event Graph of the Multiple Server Queue

Model specifics about the Multiple Serve Queue model are given below. As for the Event Graph itself, the interpretation is as follows: the Run method resets the state variables Q (number of waiting customers) and S (the number of available servers) to their initial levels and schedules an Arrival as with the Arrival Process Above. The Arrival Process, in turn, does what it always does—it increments the number of Arrivals, or, in this case, the number of customers in line, and schedules the next Arrival. Now, however, the Arrival Process is attached to another event: the beginning of a Service. There is no time delay on the edge connecting the Arrival method and the Start Service method, but there is a logical condition that says a Start Service event can only be scheduled if there is at least one Server Available (if $S > 0$). If that condition is met—a condition tested by an “if” statement within the Arrival event—then a Start Service event is put on the Event List with a zero time delay. Upon the event Start Service (which takes place instantaneously), the state variable for the number of customers in line (Q) is decremented, as is the number of available Servers (S)—since the customer being served is no longer in line, and the Server who was available to serve that customer is now serving him, and therefore no longer available—and an End Service event is automatically scheduled for a time t_s after the Service began. Note there is no Boolean condition, so any Start Service will automatically schedule an End Service. After the service time, t_s , has elapsed the End Service event is executed, whereby the “S” changes state again, indicating another Server is available, and an “if” statement is used to

schedule an immediate Start Service Event if there are any customers in line— $Q > 0$, as indicated by the condition on the scheduling edge connecting those two Event nodes.

B. COMPONENT BASED PROGRAMMING AND SIMKIT

An application of the programming concepts discussed to this point is achieved in Simkit, the simulation software package developed by Buss at the Naval Postgraduate School (NPS). The following discussion is derived from the full treatment of the subject in *Component-Based Simulation Modeling* [Ref. 19]. Simkit "uses Event Graphs as [its] underlying methodology", which means it is a DES package. Simkit is also "component based," which essentially means it extends the basic idea of object oriented programming. Programming in components is very much like programming with objects, but one level higher. Where objects are the fundamental building blocks of an object-oriented program, components, which contain objects within them, are the building blocks of more complex inter-connected programs or simulations. Within any single program or class or component, action is executed directly within the body of the program. For components to interact with other components there must be some means of connectivity, even though the component programs themselves are encapsulated. Simply stated, certain components within a simulation (for example a Patrol Boat Controller, someone who coordinates the movements of Patrol Boats) need to be able to "see" or "hear" the actions of certain other components (for example, the Patrol Boats a Patrol Boat Controller controls) in order to execute their functions within a simulation at the proper time or sequence. Writing long, complex chunks of code to connect components explicitly is not only tedious, but undermines a central premise of the object-oriented design, which is that of encapsulation. What is necessary is the "coupling" of different components, a means of allowing components to interact or respond to each other as required without linking components too directly—this "loose coupling" is a concept being developed at NPS and is the distinction between, as Buss puts it "simply designing with components and Component-Based design."

Loose coupling is achieved within Simkit in two ways, one of which is inherent to Java, and the other of which is specific to Simkit. Java includes a *PropertyChangeEvent* and *PropertyChangeListener* interface that signals or "fires" a change in state for a

variable upon the execution of a certain action within a program—for example, the addition of a customer to a queue in a service model fires a change in the size of the queue. The change event is "heard" by any object or class that has been registered as a change listener to that event. Simkit uses this feature of Java to keep track of the state trajectories throughout a simulation run so that statistics can be gathered and analysis performed. To that end, the objects generally registered as listeners to `PropertyChangeEvent`s are statistics-gathering objects or programs called "SimpleStats" objects in Simkit.

The second kind of coupling or listening pattern that is unique to Simkit is called the *SimEventListener* pattern, which is especially useful when it is necessary to know when an event has occurred. In fact, it is the *SimEventListener* pattern that allows a Simkit component based program to function at all, since it is that pattern that causes follow-on events to be scheduled upon completion of current events, thereby populating the Event List and generating the Discrete Event Simulation. "The *SimEventListener* pattern involves an event that has been executed by the Event List. It consists of the source of the event (the *SimEntity* [the Simkit specific object or class of objects] that scheduled it) multicasting the same *SimEvent* to registered *SimEventListeners*" [Ref. 19]. Hence, the capacity for an object to hear the event of another object is achieved through registration, as with the *PropertyChange* construct, but in the *SimEventListener* case, far more can be accomplished after the event is fired, because notification, in this case, also effects the execution of further actions within the listening class[es] or component[s]. The details are as follows, but the key point to be made is that the *SimEventListening* pattern is what allows a Simkit simulation's components to interact simply through the transfer of information from one entity to another, rather than through explicit commands issued from one entity to another: it is the glue of loose-coupling.

The *SimEventListening* pattern manifests the DES process described above as follows. Events, or specifically "SimEvents," are placed on the Event List when a component object or class (a *SimEntity*) within the simulation executes the method *waitDelay(String, double, object)*. The "String" within the method invocation is the name of the event to be placed on the event list. The name of the event that shows up on the list has a precise correspondence to the name of the method that executes it; the

execution method is the name with a "do" in front of it—for example, an "Arrival," as above, is actually executed within the code by actions programmed within a method called "doArrival." The "double," a number, is the time until the named event will occur—which is to say the time from the moment at which the *waitDelay* method is called or executed. The "object" argument within the method invocation (which is not required) allows specification of exactly which *SimEntity* is scheduled to undergo the scheduled event. When the named event is executed, any other component within the simulation that is both registered as a *SimEventListener* to the component that has executed the event and has an execution method with the same name (with the "do" attached) as the method being executed, will take whatever action is indicated in its own "do" method. If an object was specified in the *waitDelay* scheduling method, then that object is passed along to all listeners, thereby allowing the listeners to act upon or react to that object.

An example of this pattern from the Force Protection simulation is the way in which contacts are detected and evaluated. A contact's arrival into the river of the simulation is effected by a *doArrival* method. Within the *doArrival* method a contact is "started" which means a *doStartMove* event is scheduled via the *waitDelay* method. The actual call is something like: *waitDelay("StartMove",0.0,contact)*. Once the starting maneuver event is executed, components registered to hear it, that also have a *doStartMove* method, will be passed a reference to the contact that has started its move. In particular, the component that is monitoring all river traffic (the *PBController*) now has the capacity to know everything about the contact that can be known through invoking the contact's native methods—to include its position and motion vector. With that information the *PBController* can determine whether or not the contact is a threat, and whether or not a Patrol Boat needs to be dispatched to intercept it. How the evaluation and interception occur is largely the topic of chapter three; the point to be demonstrated here is that the loose coupling of the contact with the *PBController* is made possible through the *SimEventListening* pattern, and that loose coupling is sufficient to the task of generating the event list of a DES.

C. MULTIPLE SERVER QUEUES AND MOVERS

The above basic Event Graphs and the Event-Based programming segments they depict represent the critical conceptual building blocks of the PatrolBoat simulation. The foregoing discussion described the mechanics of Discrete Event Simulation and demonstrated the tools for presenting such a simulation's abstraction. The interest here is in presenting the model types that will be brought together to form the Force Protection simulation. The driving model of the simulation is the Multiple Server Queue, presented in Event Graph abstraction above. The Server model is the driver for the Force Protection model because waterfront force protection assets are easily cast as Servers, with incoming Threats as their customers. Upon the framework of the Multiple Server Queue will be set "Server" and "Customer" objects (the PatrolBoat and Threat objects), which are kinds of Movers. Mover objects are objects or SimEntities within the Simkit package that allow for more complex actions and interactions of and between (or among) the queuing model's components. By embedding the dynamic interaction of Simkit Movers within the Multiple Server Queuing model, a reasonable approximation of waterfront Force Protection scenarios is achieved.

Law and Kelton crystallize the queuing model in terms now, at least in part, familiar: "A queuing system is characterized by three components: arrival process, service mechanism, and queue discipline"[Ref. 10; p. 95]. The first component has already been essentially described by way of the its Event Graph. The critical issue is the timing of arrivals, the question of how and when customers enter the system. In the case of unpredictable terrorist attacks, the arrivals must be random, hence the inter-arrival time (t_A) must be a random variable—which is easily set and easily changed in the input to the simulation. According to Law and Kelton

the *service mechanism* for a queuing system is articulated by specifying the number of servers . . . whether each server has its own queue or there is one queue feeding all servers, and the probability distribution of customers' service times[Ref. 10; p. 95].

The Force Protection model is slightly more complex, since it must involve service predicated upon assessment of prospective customers (i.e. threats) before they are admitted as customers. Whereas in the basic model any object in the queue is and

remains a customer, in the Force Protection model, evaluation and prioritization will intrude upon the normal service routine. The model must allow for the most obviously threatening customers to be serviced first, and allow for the interruption of a service if the current customer ceases to be a threat. In other words, the Force Protection adaptation of the basic service model is a completely dynamic one. The last component of the queuing system, queue discipline, is “the rule that a server uses to choose the next customer from the queue (if any) when the server completes the service of the current customer”[Ref. 10; p. 95]. It is already clear that the Force Protection model will use “priority” discipline in selecting the next customer—with highest priority going to the most threatening (the closest) Mover object. What is also clear is that not every service that is started will reach a “normal” conclusion simply based upon a service time. Service time determination and the means of setting customer queuing priority, are highly dependent upon the nature of Mover objects. Hence a brief discussion of movement is required.

It is clear that simulation of moving boats must be able to incorporate movement and some sort of detection: the capacity for boats to move and for operators (either in the boats or in the control tower or elsewhere) to “see” other boats on the water. Some of the “detection” necessary in the model is achieved simply through the “listening” processes within Java and Simkit described above. Other detection is less direct and dependent upon the relative position of objects within the simulation space. Simkit provides for movement and detection through its Mover and Sensor classes, which apply straightforward mathematics to allow for modeling dynamic, interactive systems like the waterfront scenario being modeled herein. Movement is easily described over time as a function of speed and direction (i.e. a velocity vector). However, because discrete event simulations do not move forward in time steps but rather event steps, modeling movement in such simulations introduces the challenge of keeping track of the position of an object that is constantly changing between events. This challenge is met through an “implicit state” of the state variable.[Ref. 20] The implicit state notion rests upon the simple fact that position is a function of velocity multiplied by time: “a moving entity starts its move at some initial position $x_0 + (t - t_0)v$. Equivalently, the location of the entity s time units after it began its movement is $x_0 + sv$ ”[Ref. 20]. In other words, if an

object's initial position and velocity—which of course means both speed and direction, since it is a vector—are known, along with its start time, then its position at any known time after the start time can be calculated at any time, given there has been no change in its velocity. The value of the implicit state to the discrete event simulation environment follows: changes in an entity's movement are simply events or methods that determine the future position of the entity until its next movement event. The implicit state is highly leveraged within Simkit in the detection process, implemented through "Sensor" objects by way of third party "Referees" and "Mediators" that prevent the Sensors from knowing ahead of time when a Mover is going to enter its range.

The Patrol Boat movement application is somewhat different from the sensing problem, but it, as well, relies upon the implicit state. A Patrol Boat's job, particularly in the line of sight situation on the Sub Base Bangor riverfront, is not one of sensing, but of evaluating and intercepting. Evaluation is easily accomplished and does not involve movement per se, but interception of threats does. In reality, intercept points are calculated using something like a maneuvering board or its electronic equivalent, by determining a target's relative speed and direction over time and then adjusting one's course and speed in order to intercept. The development of a target's relative behavior with time is a time-stepped phenomenon and therefore not applicable to the discrete event construct. Nonetheless, given a target's initial position and vector—which is to say that target's position and vector at the instant of a specific event—the implicit state allows for the calculation of an intercept point which will remain valid at least until the next event of the threat in question, as long as an intercept speed is defined. If an intercept speed is not defined, the basic motion equation, $x(t) = x(0) + v * t$, will end up with two unknowns, velocity and time. The ability to calculate an intercept point is essential to the simulation of the Patrol Boat movement, since interception is precisely what defines the "service" within the dynamic service model being developed. The mathematics underlying the intercept algorithm—code available in Appendix A—is as follows, for an intercept by Patrol Boat, PB, of a threat, C, with initial conditions at a *doStartMove* event of C that is heard by PB:

Givens:

$$\text{PB initial position: } \begin{bmatrix} X_{pb} \\ Y_{pb} \end{bmatrix} \quad \text{PB velocity: } \begin{bmatrix} PBx \\ PBy \end{bmatrix}$$

$$\text{Threat initial position: } \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \quad \text{Threat velocity: } \begin{bmatrix} Cx \\ Cy \end{bmatrix}$$

Using the basic motion equation for both the Patrol Boat and the threat, set their positions at time, t , equal to solve for the time at which they will be in the same place, given a Patrol Boat intercept speed:

$$(1) \quad \begin{bmatrix} X_{pb} \\ Y_{pb} \end{bmatrix} + \begin{bmatrix} PBx \\ PBy \end{bmatrix} * time = \begin{bmatrix} X_c \\ Y_c \end{bmatrix} + \begin{bmatrix} Cx \\ Cy \end{bmatrix} * time$$

$$(2) \quad \begin{bmatrix} X_c - X_{pb} \\ Y_c - Y_{pb} \end{bmatrix} = \begin{bmatrix} PBx \\ PBy \end{bmatrix} - \begin{bmatrix} Cx \\ Cy \end{bmatrix} * time \quad \text{let: } \begin{matrix} [X_c - X_{pb}] = dx \\ [Y_c - Y_{pb}] = dy \end{matrix}$$

$$(3) \quad time = \left[\frac{dy}{PBy - Cy} \right] = \left[\frac{dx}{PBx - Cx} \right]$$

$$(4) \quad \frac{[PBx - Cx]}{[PBy - Cy]} = \frac{dx}{dy} \Rightarrow dy(PBx) - dy(Cx) = dx(PBy) - dx(Cy)$$

Now, with the unknown, $time$, removed, the necessary intercept velocity can be obtained by using straightforward vector resolution. First, for simplification of terms, since the Threat velocity and both vessels' initial positions are known, we can define a constant, $k = dy(Cx) - dx(Cy)$. Also, since the Patrol Boat's velocity is in part what is being solved for—that is, its direction or course to intercept—we must define an intercept speed, S , the magnitude of the intercept vector, so that the Pythagorean theorem can be invoked, leaving as a single unknown either of the intercept vector's component vectors. Here, as in the algorithm within the simulation coding, the X component is solved in terms of the Y :

$$(5) \quad |PBx|^2 + |PBy|^2 = |V_{int}|^2 = S^2 \Rightarrow PBx = \sqrt{S^2 - PBy^2}$$

Further algebraic manipulation yields:

$$(6) \quad (dy^2 + dx^2)PBy^2 + 2(dx)(k)PBy + (k^2 - (dy^2)(S^2)) = 0$$

Which can be solved using the quadratic formula , with

$$a = (dy^2 + dx^2), b = 2(dx)(k), c = (k^2 - (dy^2)(S^2)) , \text{ and}$$

$$(7) \quad PBy = \frac{-b \pm \sqrt{b^2 - 4(a)(c)}}{2(a)}$$

At this point the Patrol Boat's vector for intercept is fully determinable, but for Simkit purposes, the key value is that of time to intercept, which is obtained using (3). The time to intercept is used in the basic motion equation ($x(t) = x(0) + v * t$) for the Threat to solve for the intercept point—that point at which, at time = *time* in the future, the Threat's position will equal that of the Patrol Boat. Actually solving for the Patrol Boat's vector is unnecessary within the simulation program, since an order to move to the intercept point will necessarily result in that vector's generation—in fact, Simkit will provide the vector if desired, through a method invoked upon the Patrol Boat object upon its execution of the event that moves it to intercept.

It should be noted, at this point, that certain conditions make an intercept infeasible. There are three types of intercept infeasibility that are apparent in the development above. The first is infeasibility because there simply is no point at which a Patrol Boat and target's motion equations will be equal. This is indicated by an imaginary root—a negative result under the radical of (7). The second infeasibility results when a calculated intercept point, while it exists, is outside the defined problem space—e.g. an intercept point that is actually on land. The third infeasibility is the case where time to intercept, from (3), is negative, indicating that the intercept point as already been passed.

Infeasibilities notwithstanding, once the intercept point of an incoming threat has been determined, the pieces of the dynamic service model are assembled. Incoming contacts that are evaluated as threats are the customers, the Patrol Boats are the servers, and the intercept maneuvers derived from the calculation above are the services. Unlike

typical service models, customers do not stand still if they join the queue, and service times are not simply the realization of a random variable, but are rather a function of the distance separating the server from the customer at the start service. Because of possible speed mismatches, there is also the possibility of service infeasibility—i.e. the inability for a Patrol Boat to intercept at its intercept speed, given a high Threat speed—which doesn't exist in the standard service model.

The foregoing has presented the tools and modes of presentation of discrete event simulation of a service model with movement. The following chapter applies the principles herein discussed to the waterfront Force Protection problem and outlines the development of the model generated to simulate it, using Event Graphs. It also addresses the issues of verification and validation and the use of graphics, in preparation for model runs and analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

III. MODEL DEVELOPMENT

A. MODEL COMPONENTS

The components of the Force Protection simulation are, in true object-oriented fashion, made up of entities that represent the basic "artifacts" of the waterfront at SubBase Bangor—or, really, any port—and the command structures that direct them. The main objects of the simulation are boats, both Patrol Boats and other boats on the river. These objects are extensions of Simkit's *BasicMover* class, and are called *TypedBasicMovers*, to allow for differentiating, for example, patrol craft from threat craft within the code. The components of the simulation program are the classes that generate, command and move the *TypedBasicMovers*. Certain components need to know what certain other components are doing, so they are set up as *SimEventListeners* to their "subordinate" classes. The figure below is a black-box depiction of the simulation's major components and their interactions, with *SimEventListeners* connected by the bold arrows. This bird's-eye view shows the essential structure of the entire simulation model; the event graphs that follow actually reside, conceptually, within the component blocks.

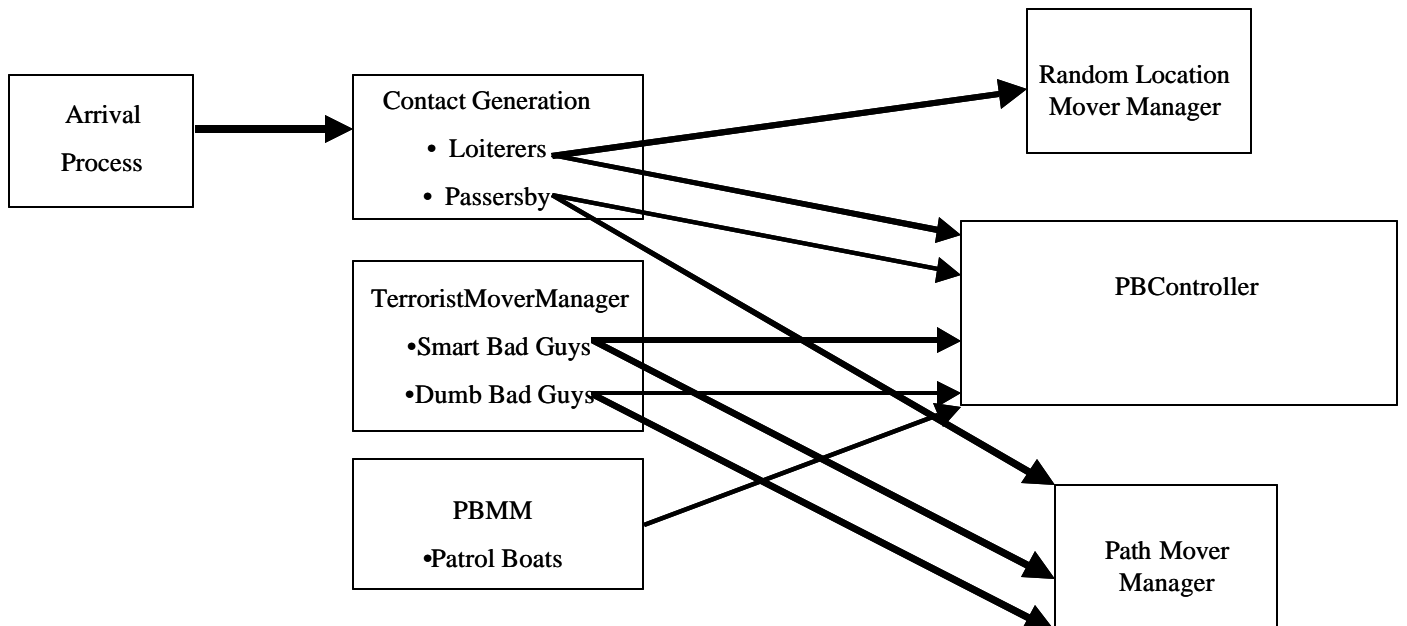


Figure 5. Major Simulation Components and Listening Pattern

The arrowheads show the direction of flow of information in the SimEventListener pattern of the simulation. For example, the Contact Generators listen to the Arrival process and the PBController listens to individual contacts, and to all Patrol Boat Mover Manager (PBMM) components. The component-based design of the simulation above is enough to describe the general processes of the model. Arrival Processes provide inter-arrival times at which different kinds of contacts are generated within a contact generation component. Each contact is assigned to a movement-managing component, dependent upon its type, and is made known to the PBController through the PBController's being made a SimEventListener to the contact—the reality simulated here is that the Arrival Process signifies a contact's arrival into the field of view of the controlling station. Patrol Boat objects are instantiated and assigned to PBMM's and are, as well, put under the supervision of the PBController. Finally, each simulation has a TerroristMoverManager component that carries out a specific kind of attack upon the Delta pier. The terrorist objects are movers like the others and are, therefore, also assigned mover managers and registered with the PBController, though the PBController is not told they are any different from any other contact on the river. The listening pattern allows mover manager components to hear movement events so as to schedule follow-on movement events while simultaneously allowing the PBController to hear those same movement events so as to respond to them.

1. Mover Managers

There are four kinds of Mover Managers implemented within the simulation: a RandomLocationMoverManager for Randomly maneuvering TypedBasicMovers—called Loiterers—a basic PathMoverManager transiting or "Passerby" TypedBasicMovers, a TerroristMoverManager for "Bad Guy" TypedBasicMovers, and a PatrolBoatMoverManager (PBMM) for Patrol Boat TypedBasicMovers. The basic mover manager components of the simulation are very straightforward but worth showing, since they demonstrate the way in which the dynamic service model is set and maintained in motion. The basic idea of all types of mover managers is that they move their constituent objects from one point to another. The differences among the different types of Managers result from differences in the behaviors of the various TypedBasicMovers. For

example, the difference between random movement and path movement is simply that a RandomLocationMoverManager generates a new random location as the next destination point for its Mover upon its arrival at a location (which results in endless movement, unless interrupted), whereas all other Mover Managers move their constituent Movers through a sequence of pre-set waypoints. In either case, the Mover—or within the context of this simulation, the TypedBasicMover—is started and then redirected upon every execution of its *doEndMove* method.

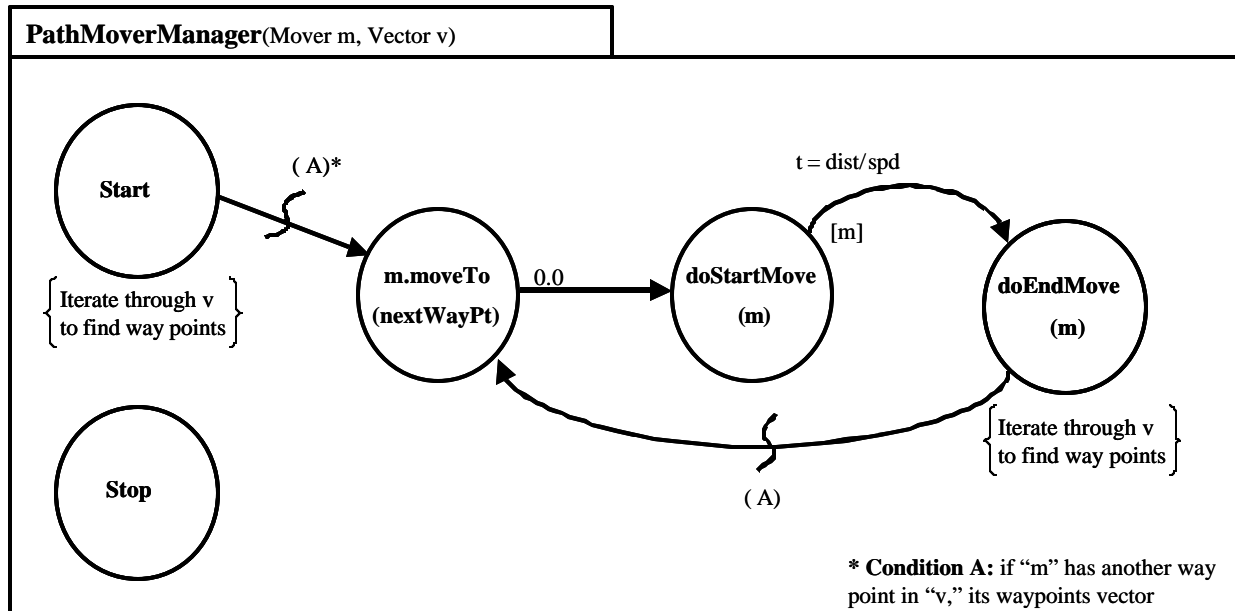


Figure 6. PathMoverManager Event Graph (major elements)

A few additional Event Graph conventions in this and the following applications need to be explained. The labeling techniques used in the above Event Graph (Figure 6) will be used in all the graphs that follow. The items within the parentheses appended to the class names on the top of the event graphs, are objects that are "handed" to the component upon its instantiation, through its constructor. Once the component has the objects, it can directly invoke methods or perform operations upon them. The names are included in the event graphs to show how direct invocations upon an object can be made within the depicted classes. For example, the fact that the PathMoverManager is handed a reference to the Mover it will control ("m") and that Mover's list of waypoints ("v") allows the PathMoverManager to invoke the Mover's *moveTo* method directly (*m.moveTo*), and to iterate through its Mover's waypoints. Conditions on arcs are labeled

and then explained below each graph to save space. As usual, changes in state are indicated in curly braces below applicable methods. Herein, important processes that are executed within a method will also be indicated within those curly braces. The brackets below certain scheduling edges indicate a parameter that is being passed from a scheduling method to the method being scheduled—this corresponds either to a direct method invocation that requires the parameter or to the object instantiated within the *waitDelay* construct discussed above. Perhaps most importantly, it must be understood that the event graphs shown here are not comprehensive in that they do not show every single method or capture every action or every change of a state variable within the classes or components they represent. The graphs are designed (as always) to capture the main flow of events in the DES while limiting the complexity of their presentation.

As for the details of the PathMoverManager event graph, the most important thing to note is that the key event in the class is the *doEndMove* event, because it is here that, once the manager has been started, it takes all follow-on actions. This same idea is elaborated in the PBMM. The graph also visualizes the SimEventListener pattern, in that the *doEndMove* method of the PathMoverManager is executed upon its "hearing" the same event executed by its Mover. Once the PathMoverManager has been added as SimEventListener to its Mover—which actually happens in another class, as will be seen—it will hear and therefore be able to react to any of its Mover's events within its own event of the same name. Specifically, then, after the PathMoverManager has been started, through an invocation of its *start* command (shown as a method for ease in following the flow of events), it checks to see if its Mover has any waypoints and, if so, it directs the Mover to its first waypoint. The *m.moveTo*(some point) command results in an immediate *doStartMove* for the mover and a scheduled *doEndMove* for the same mover with a time delay equal to the time it will take for the Mover to move from where it is to its first waypoint, based upon the simple motion equation. The PathMoverManager does not, in fact, have a *doStartMove* method within it; it merely effects movement through the *moveTo* command. The method has been included within the event graph to make the movement process more apparent. When the Mover executes its *doEndMove* method, upon arrival at a waypoint, that event is heard by the PathMoverManager, which then iterates through the waypoint vector again and issues

another *moveTo* event, if there are more waypoints in the vector. Note that if there are no more waypoints, the PathMoverManager simply does nothing, hence the Mover will remain at rest. The *stop* command (again shown as a method), therefore, is merely included to allow a direct intervention in the pre-set movement of a Mover.

2. Contact Generators

There are two kinds of contact generators in the model, each corresponding to a specific kind of TypedBasicMover. One is a LoitererGenerator, which generates TypedBasicMovers of the PlatformType "Loiterer," and the other is a PasserbyGenerator, which generates TypedBasicMovers of the PlatformType "Passerby." Either type of contact is generated at a random "Y" position—the Y-axis being the East-West axis or "across" the river from the pier—on either end (north-south axis) of the river zone, and given a uniform random speed between zero and thirty-five knots. The only difference in the generators is the kind of Mover Manager to which it assigns its respective TypedBasicMover. A Loiterer is a randomly moving contact, meant to represent a fishing boat, or a recreational craft moving around in the vicinity of the pier. Consequently Loiterers are assigned to a RandomLocationMoverManager that simply moves its Loiterer randomly from point to point within the river bounds. Loiterers will often cross into the exclusion zone and be evaluated as threats that need to be intercepted. They are distractions for the Patrol Boats, but since there is no way of knowing which contacts might be Terrorists, Loiterers must be taken seriously. This is an important feature of the model, because the primary goal of the simulation is to come up with some way of effectively preventing a terrorist attack without knowing for sure whether or not any given boat may contain a terrorist. A Passerby is a contact that moves in a straight line from one edge of the river boundary to the other, passing by the sub-base. A Passerby is a transiting craft that is generally not going to drive inside the exclusion zone in the river and consequently will rarely be evaluated as a threat, but its randomly assigned Y-position is allowed to fall inside the exclusion zone by up to fifty yards, so one may well end up being intercepted as a threat upon occasion. Since the contact generators are very similar, only the LoitererGenerator event graph is provided:

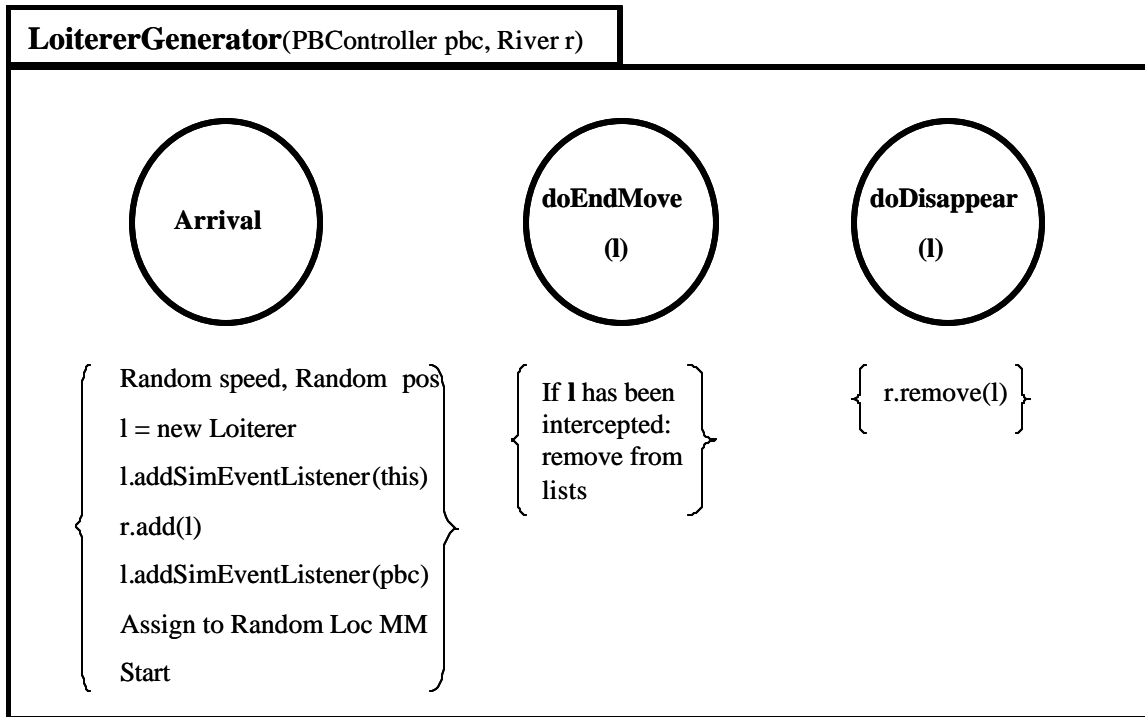


Figure 7. LoitererGenerator Event Graph (major elements)

An element shown for the first time in this event graph (Figure 7) is the *River* object, which is part of what allows the Patrol Boat simulation to be displayed graphically. The simulation's "river" is passed into the LoitererGenerator and other boat components of the simulation so their movement may be seen as the simulation runs. The graphical display of the simulation will be discussed below, and has nothing to do with the way the simulation, per se, works. Since LoitererGenerator is a SimEventListener to an ArrivalProcess—not shown in itself, but the same as shown in Chapter Two—it has its own *doArrival* method, which is executed upon an Arrival event. Hence "generation," which is merely instantiation of an object of a specific type, takes place upon an Arrival. Upon instantiation within the *doArrival* method, the new Loiterer object is assigned a mover manager, made known to the PBController, and started upon its randomly moving way. The *doEndMove* method is heard from the instantiated Loiterer object, since the LoitererGenerator is a SimEventListener to its Loiterer—the Generator is the "this" of "l.addSimEventListener(this)." Upon a Loiterer's execution of

an EndMove after it has been intercepted—a condition known through a Boolean method invoked upon a TypedBasicMover—the LoitererGenerator essentially removes all records of it, except its graphical depiction. The graphical depiction of the Loiterer is not removed from the simulation's display panel until its *doDisappear* method is invoked a short time after its interception.

3. TerroristMoverManager

The TerroristMoverManager component is an extension of the PathMoverManager class, specifically tailored to control terrorist or "Bad Guy" movement. It is described separately to show more clearly how the main event of the simulation—the terrorist attack—is effected. It is the component that executes terrorist attacks, by moving "Bad Guy" objects toward the High Value Unit (HVV). It is passed a TypedBasicMover of PlatformType either "Smart Bad Guy" or "Dumb Bad Guy," with its waypoints and a time until attack. A Smart Bad Guy has two waypoints in its waypoint vector: the first with the same Y-position as its initial starting point and an X-position that is the same as the X-position of the HVV, and the second set equal to the position of the HVV. Hence, a "Smart Bad Guy" transits parallel to the exclusion barrier at a low speed (like a Passerby) until directly opposite the pier, and then turns ninety degrees toward the pier and accelerates to maximum target speed. A Dumb Bad Guy is more direct. It is started at either end of the river, and simply drives straight toward its single waypoint, which is the location of the HVV. Both kinds of Bad Guy have the High Value Unit's location as an ultimate destination, which differentiates their movement from other TypedBasicMovers.

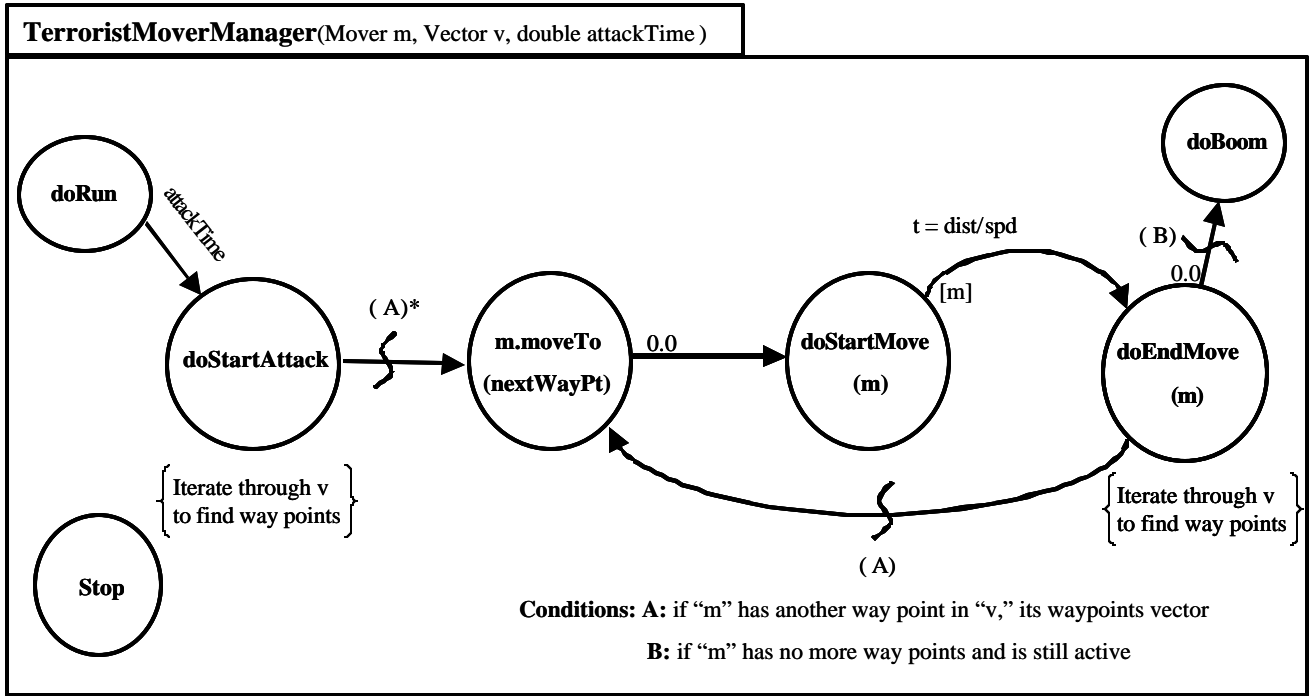


Figure 8. TerroristMoverManager Event Graph (major elements)

The TerroristMoverManager (Figure 8) component or class is very similar to the other Mover Managers, except that its movement is not activated directly by a "start" command, but rather by a *doAttack* method scheduled with a *waitDelay* method within the class upon the execution of the *doRun* method at the simulation's start. Additionally, the TerroristMoverManager class, fires a *doBoom* event, indicating a terrorist has achieved its goal of getting to the HVU, if upon a *doEndMove* event it has no more waypoints and is still "active"—an active threat is one which has not been intercepted. The *doBoom* event is the source of the "terror" statistic defined in chapter Four.

4. PBMM

The PBMM—short for Patrol Boat Mover Manager—class does precisely what it says: it manages the movement of TypedBasicMovers of PlatformType "Patrol Boat." In reality terms, a PBMM can be thought of as the captain or operator of a Patrol Boat. In Simkit terms, it is an elaboration of the basic PathMoverManager that controls the movement of Passersby and Bad Guys of either kind. The PBMM includes methods

specific to a Patrol Boat object, most importantly an "Intercept" method—which is not a "do" method, because it need not be heard. The PBMM's normal mode is the patrolling mode: it moves a Patrol Boat through the waypoints of its set patrol pattern until it reaches the end, and then starts it over again, ad infinitum. It "knows" that its Patrol Boat is in patrolling mode as long as it has not been assigned a target, which it checks upon the execution of each of its Patrol Boat's *doEndMove* methods. If the Patrol Boat has ended a move without an assigned threat, the PBMM keeps moving it through its patrol pattern. The patrol pattern is only broken when the PBMM is given directions by the PBController to intercept a threat contact. Upon the direction to intercept, the PBMM stops the patrolling pattern and moves its Patrol Boat TypedBasicMover to the intercept point it has been handed by the PBController. Upon the execution of its TypedBasicMover's *doEndMove* method, when the Mover is *not* patrolling—indicated by its having an assigned threat object—the PBMM schedules a *doEndIntercept* method with a zero time delay, which is "heard" by the PBController through the SimEventListener pattern. At the end of an intercept move, the PBMM will either be immediately re-assigned to intercept another threat or be released to re-commence its patrol cycle.

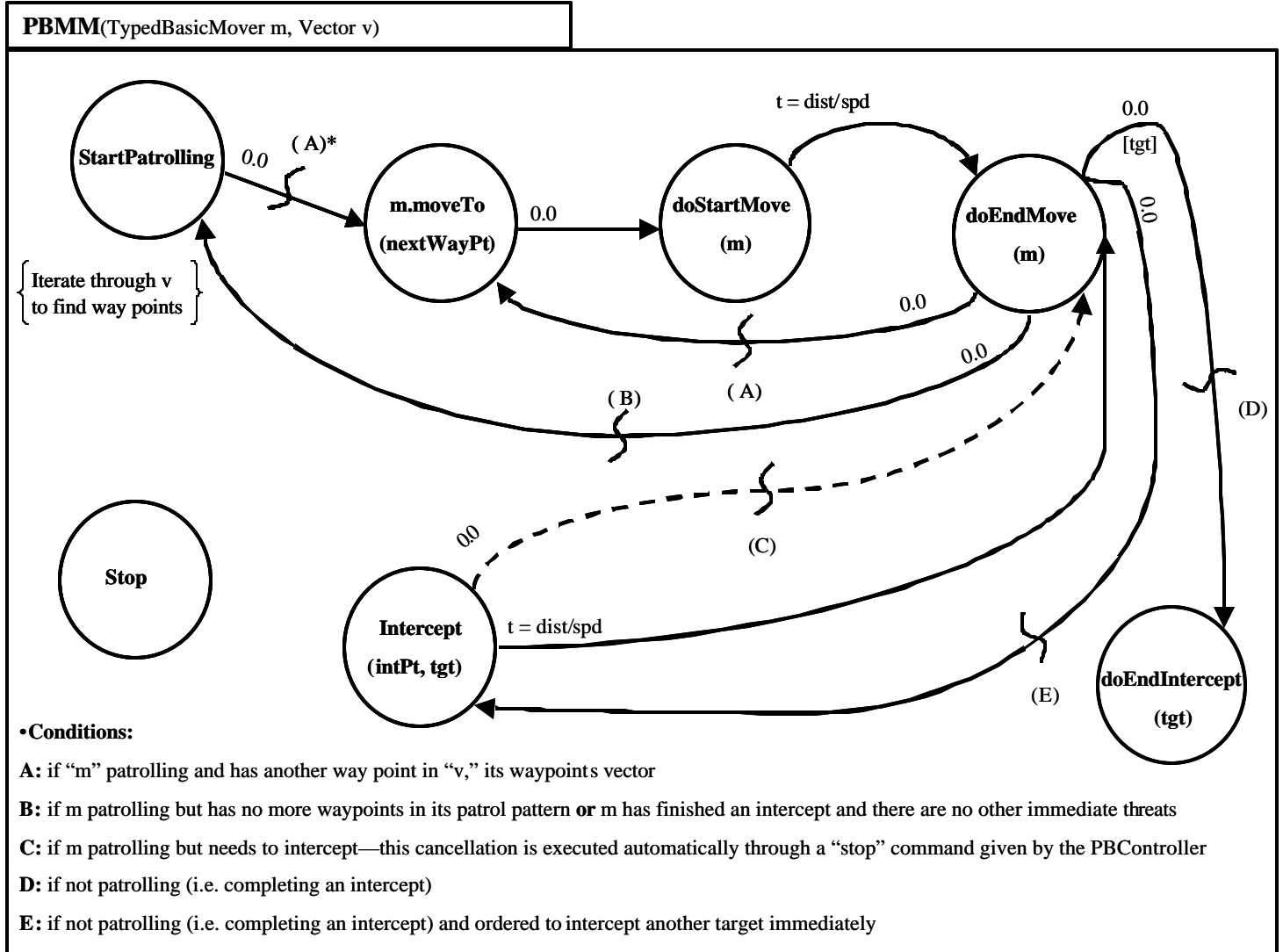


Figure 9. PBMM Event Graph (major elements)

As with the PathMoverManager, the *doEndMove* is the critical event of the PBMM, since that is where it does most of its evaluation and scheduling. When a Patrol Boat is patrolling, its movement is indistinguishable from that of other Movers, except that it will re-start at the end of its waypoints as opposed to other Movers that just stop. This restart is accomplished through additional condition labeled "B" in the graph (Figure 9). What is very different about a PBMM relative to a PathMoverManager is that its normal movement cycle can be interrupted by the PBController through its invocation of the PBMM's "Intercept" method as shown. The Intercept method receives the target to be

intercepted and its location—this is the boat's operator being vectored to a specific target by the contact/strike coordinator. It uses the intercept point location to invoke the *moveTo* method on its Patrol Boat object and retains (through a setting method) the target reference so it can schedule a *doEndIntercept* upon that specific target, once the intercept move is complete and assuming nothing changes after the first command to intercept. The cancellation—represented by the dotted line—from the Intercept method to the *doEndMove* is not explicitly executed in the PBMM, but is included to show that an invocation of the PBMM's Intercept method automatically stops the PBMM's Patrol Boat thereby canceling its previously scheduled EndMove, to allow redirection before completion of a planned maneuver. If a PBMM's Patrol Boat gets to an EndMove event with a reference to a target still active, the PBMM, upon hearing the event, will schedule a *doEndIntercept* event with a zero time delay, passing along a reference to the target which has been intercepted to the PBController through the SimEventListening pattern. Movement after an EndMove event constituting an intercept is actually controlled via the PBController—which decides whether or not it still needs the PBMM for intercepting—but the possible movements after such an EndMove and the logic that effects them are depicted in the PBMM event graph for sake of convenience and clarity.

5. PBController

The PBController is the coordinating component of the simulation. It is the class that monitors both contacts and Patrol Boats and directs Patrol Boats based upon its evaluation of contacts within the river. In reality terms, the PBController is like a combined contact and strike coordinator. It tracks contacts and maintains what amounts to a status board of all activities on the river: lists of all contacts, all threats, all available Patrol Boats, and all intercept assignments. It also determines whether or not a contact needs to be intercepted, which Patrol Boat will intercept it, and where that Patrol Boat needs to go to intercept—the algorithm for the intercept problem developed above resides within this class. Because the PBController listens to all other components within the simulation and directs all interaction between Patrol Boats and contacts it is the component within which almost all of the simulation's statistics are gathered.

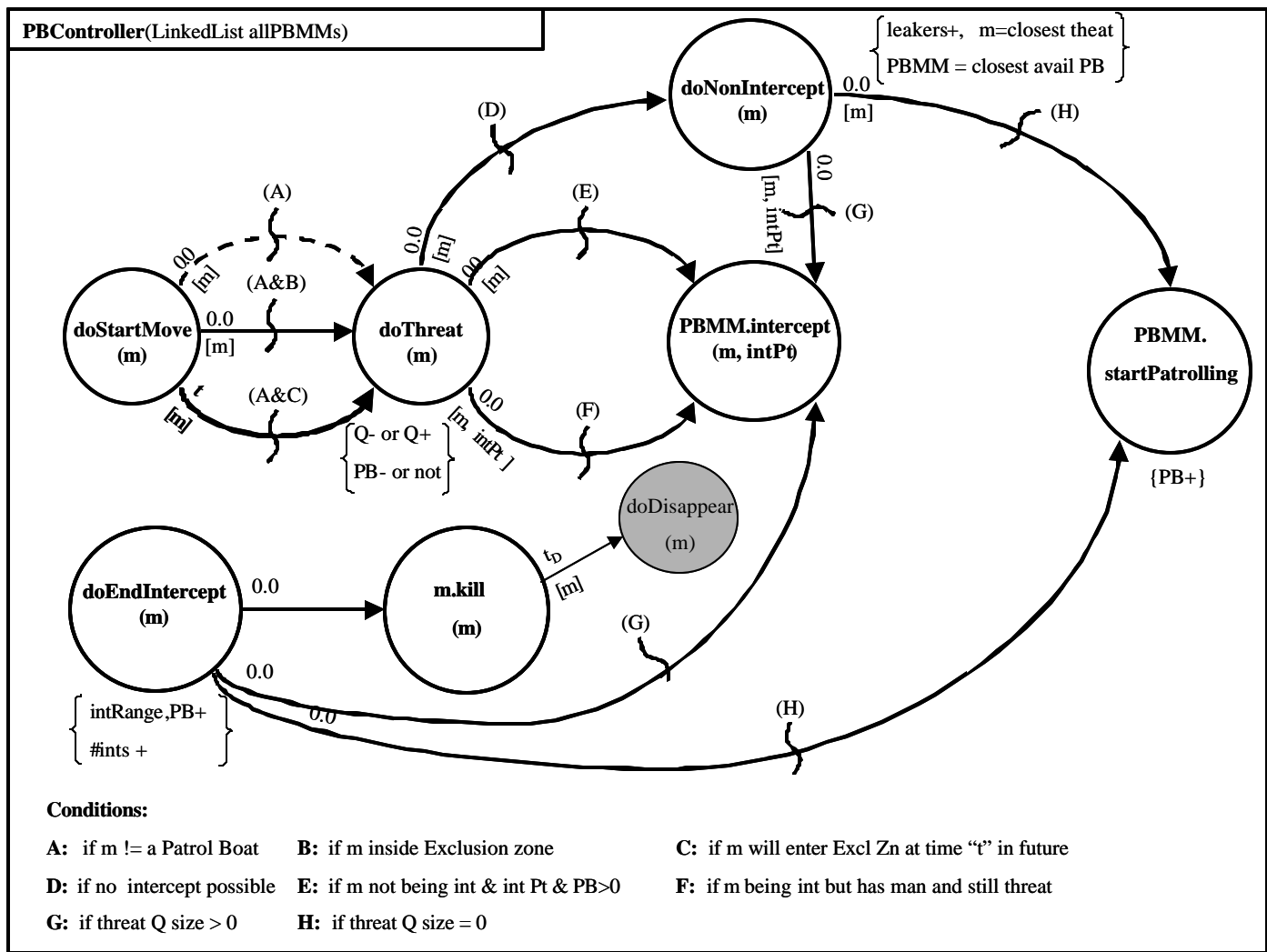


Figure 10. PBController Event Graph (major elements)

The PBController (Figure 10) is handed the list of its PBMM's for a given simulation scenario. It is also a SimEventListener to its PBMM's and to every other contact that is generated within the simulation, as has been shown. This means that the PBController can hear any SimEvent fired by any boat object in the simulation within its own methods of the same names. The event that most matters is the *doStartMove* event of any non-Patrol Boat, because upon the execution of a StartMove event, a contact's movement information for its next leg of motion is available, through its various methods, to any object holding a reference to it. This is how the PBController evaluates a contact to see if it is a threat. If a contact is a threat, which means it is currently, or will at some point be, inside the exclusion zone of the river, then the PBController schedules a

doThreat event. It is scheduled immediately if a contact is already inside the exclusion zone upon its *StartMove*—as is the case with a contact that is instantiated inside it or one which maneuvers inside after previously entering but before interception—or with a *waitDelay* equal to the time at which the contact will cross the line, as determined by the basic motion equation. The canceling edge means the first step taken by the *PBController*, whenever any of its contacts maneuvers, is to cancel previously scheduled Threat events for that contact. This allows for the re-evaluation of a maneuvering contact and is ignored if no Threat events have been scheduled for the contact that has started a move.

The *doThreat* method is perhaps the most important of the entire simulation. It takes in all the motion information for a contact and applies—through another method not shown, called *getIntercept*—the algorithm developed above to determine if an intercept of an identified threat is possible at the intercept speed set for a given simulation scenario. If not (condition D in Figure 10) it schedules—actually, again, through the *getIntercept* method—a *doNonIntercept* method with a zero time delay. If so (conditions E or F), it determines the available Patrol Boat closest to the threat that can intercept it and invokes its *PBMM*'s *Intercept* method. If there are no feasible available Patrol Boats, it adds the threat to the threat queue—this is why the state variables change in different possible ways in this method. It should be clear that a *doThreat* method within the *PBController* that yields a feasible intercept and an *Intercept* method within a *PBMM* will take place at the same simulation time, given there is a Patrol Boat available. What may not be quite as obvious is that these coordinated actions represent the Start Service event of the dynamic service model.

The *doEndIntercept* method of the *PBController* represents the End Service of the dynamic service model, but is not directly connected as in the general case shown in Chapter Two, because it is actually an event fired from the *PBMM*, and therefore only heard, rather than executed by the *PBController*. The *PBController*, the ultimate micro-managing middle man, only schedules and records completion of service; it doesn't actually perform any! Nonetheless, the *doEndIntercept* method accomplishes the same general tasks as an *EndService*: it takes statistics on service performance, and schedules follow-on services if necessary. If a server (Patrol Boat) is "idle" it is sent back into its

patrolling pattern. The tone of the dynamic service model is apparent in the *kill* method invoked upon any threat whose "service" is complete. The method actually resides in the TypedBasicMover class and simply stops a TypedBasicMover dead in its tracks and schedules a *doDisappear* with a pre-set *waitDelay*. The delay is set to allow the graphical display to show a contact has been intercepted or killed before it disappears. Upon an intercept, the intercepted threat turns a dark color—magenta for non-Bad Guys; black for Bad Guys—and its image is held until the execution of the Disappear method, at which time the object disappears from the screen. The Disappear method is highlighted in the event graph to show it is actually a follow-on action performed outside the scope of the PBController.

6. BreachSensor

This component of the simulation is not in the block diagram above, because, although not in the SimEventListening pattern, it does serve an important purpose and does incorporate the FirePropertyChange construct. BreachSensor is an extension of the Simkit BasicSensor. It is used in this case simply to gather a statistic on the number of contacts that get inside a "breach range" or standoff range from the pier. In other words, it is not really being used as a sensor but rather as an indicator of a "breach" event. The breach range is set as the range of the sensor and can be easily varied. The goal is to provide an additional measure of Patrol Boat performance to the measure of whether or not a terrorist has gotten all the way to a HVU. The idea is that any contact within a short distance from a moored submarine represents a degree of failure in force protection, even if the threat is intercepted.

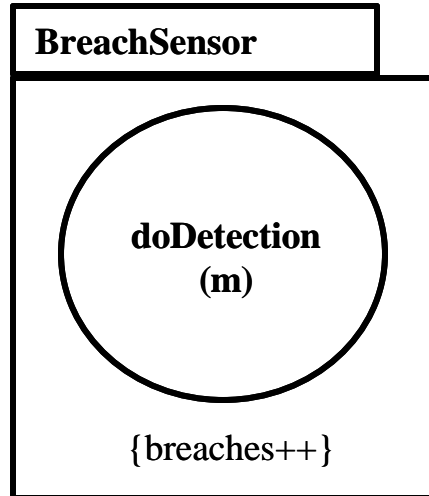


Figure 11. BreachSensor event graph

The MyBasicSensor class (Figure 11) listens to the *doDetection* method of its Referee object—an object used with Simkit sensing to coordinate targets and sensors that are registered with it via a Mediator—and upon execution of the method, it fires a PropertyChanged, simply incrementing the number of breaches for the run.

B. OTHER CONSIDERATIONS

1. Face Verification and Validation: Assumptions

An explanation of component design and interaction demonstrates how the simulation model is constructed, but model results are dependent upon what assumptions are made about model inputs, and whether or not those assumptions are reasonable and properly implemented. Bratley, Fox and Schrage define verification as "Checking that the simulation program operates in the way that the model implementer thinks it does: that is, is the program free of bugs and consistent with the model"[Ref. 21; p. 8]. Verification is essentially a coding check; it ensures that the model's constituent algorithms are doing their calculations correctly. Simkit provides an excellent tool for verification in its "verbose" mode. When Simkit's Schedule is set to *verbose*, it lists the current Event being executed in the DES, along with the Future Event List and the names, positions and speeds of all instantiated Movers. This allows the programmer literally to step through a simulation run and ensure Movers are behaving as they are designed to behave. Of particular importance in the simulation is the Intercept move. The

underlying algorithm was verified both through the Event List output—that showed proper speed changes and that the Patrol Boat and target did in fact end up at precisely the same point—and by an independent spreadsheet implementation. The model's graphical display was also an excellent tool for model verification. A picture tells a thousand words, and a moving picture—in fact, a picture of Movers—exposes the functionality of a thousand lines of code. With graphics one can easily and literally achieve "face verification" just by checking to see if everything "looks about right."

Validation is a more elusive goal. Department of Defense Directive 5000.59, *DoD Modeling and Simulation (M&S) Management*, defines validation as "the process of determining the degree to which a model or simulation is an accurate representation of the real-world from the perspective of the intended uses of the model or simulation"[Ref. 22]. Bratley et al. define validation as "Checking that the simulation model, correctly implemented, is a sufficiently close approximation to reality for the intended application . . . no recipe exists for doing this"[Ref. 21; p. 9]. Probably the best that can ever be achieved, particularly in a data-less, exploratory model like this one, is "face validation." Nonetheless, a few points can point toward accuracy. Firstly, the model is designed to allow for changes in model inputs, so that it is robust to varying scenarios and therefore can be adjusted to achieve increasing levels of validity. Key parameters can be easily changed—and are, for analysis purposes, in the next chapter—to get a range of outcomes. However, certain vital behavioral patterns are held constant and need to be justified by being held up to reality. First are the ArrivalProcesses for the contact generators. The Arrival Processes are modeled essentially as Poisson processes. As Ross points out, the Poisson process, a counting process which has exponentially distributed interarrival times, can be used in situations where a system starts empty and interarrival times are independent (the assumption of "independent increments"[Ref. 23; p. 251]). The vital and unique[Ref. 23; p. 238] attribute of the exponential distribution that makes its application to the arrival process very reasonable is its "memory-less" property; the fact that at any point in time, the expected time until the next arrival is the same as it had been at time zero:

The assumption of stationary and independent increments is basically equivalent to asserting that, at any point in time, the process

probabilistically restarts itself. That is, the process from any point on is independent of all that has previously occurred . . . In other words, the process has no *memory*, and hence exponential interarrival times are to be expected.[Ref. 23; p. 256]

In the model, inter-arrival times of the two Arrival Processes are generated from two different Gamma distributions. There are no specific data of river traffic for the area adjacent to the Delta pier, so the model implements a more general form of the exponential arrival rate recommended by Ross and Law and Kelton[Ref. 10; p. 300]. Both Arrival Processes in the model—for Loiterers and Passersby—use Gamma distributions with shape parameters equal to one, which *are* exponential distributions, but could be adjusted easily if future data indicated a more appropriate distribution shape within the Gamma family. The model is meant to stress the capacity of Patrol Boats to respond to threats in very high traffic density (a conservative approach) so inter-arrival rates are relatively high particularly for Loiterers. The mean arrival rate for Loiterers is 0.125 (eight arrivals per hour), and that for Passersby is 0.5 (two arrivals per hour). These values are set to give "conservative" estimates of Patrol Boat performance, and are held steady while the parameter space of Patrol Boat configurations is explored.

A second key characteristic of the model is how a contact is evaluated as a threat. As it stands, a threat—which is to say, a contact that will be intercepted if possible—is defined as any contact that crosses into the exclusion zone, which is established, in reality, by the Coast Guard and marked with yellow buoys. According the algorithm, a Patrol Boat will never move to intercept a contact until it crosses the line, and when it does move to intercept it will always move at the same speed, regardless of contact characteristics. The criterion for responding only with a violation of the exclusion zone was established based upon discussion with the CSG-9 security officer, and the uniform intercept speed was established based upon the fact that all threats must be treated as real threats.

Another implicit assumption of the model is that the PBController has perfect information about certain characteristics of every contact on the river. While it cannot know a contact's type—since that would be cheating—it does know everything about its location and movement. It "sees" and can direct an intercept of all contacts immediately

upon their entering the two-mile by one-mile rectangle that represents the operational zone around the pier. This assumption is supported by the fact that, in reality, from the control tower on Delta pier there is a full view well beyond the zone depicted in the simulation, and the entire operational area can be surveyed continuously. In other words, this is a line of sight situation, where it is reasonable to leave out delays in detection time. Certainly, there is a case to be made for the low visibility, bad weather situation, but that is not taken into account in the model. The capacity to generate an intercept point immediately is actually more accurate than it may at first appear, since a maneuvering Patrol Boat, with line of sight, is capable of simply driving at a contact, making whatever course adjustments are necessary to achieve an intercept. Were there truly the necessity for a maneuvering board type calculation to be done before an intercept course could be determined, the instantaneous solution would be unreasonable, but in this line of sight case, the movement is reasonable.

2. Graphics

The use of graphics in presenting the Force Protection simulation has already been mentioned in the description of component event graphs and in its value for model verification. Two more points need to be mentioned. First, the basic mechanics of the graphics, and second the value of graphics in the use of the simulation.

There is one class within the simulation package, *Patroller*, that implements Java Swing. That class is called the River, as mentioned above. Within River, directions are given for drawing anything that should be visible on the river, and methods are included for adding TypedBasicMovers to the river. In particular, the coordinate axes, the delta pier, the exclusion zone barrier and the breach range are permanently displayed on the panel (Figure 12). TypedBasicMover objects are displayed after they have been added to the panel until they have been removed.

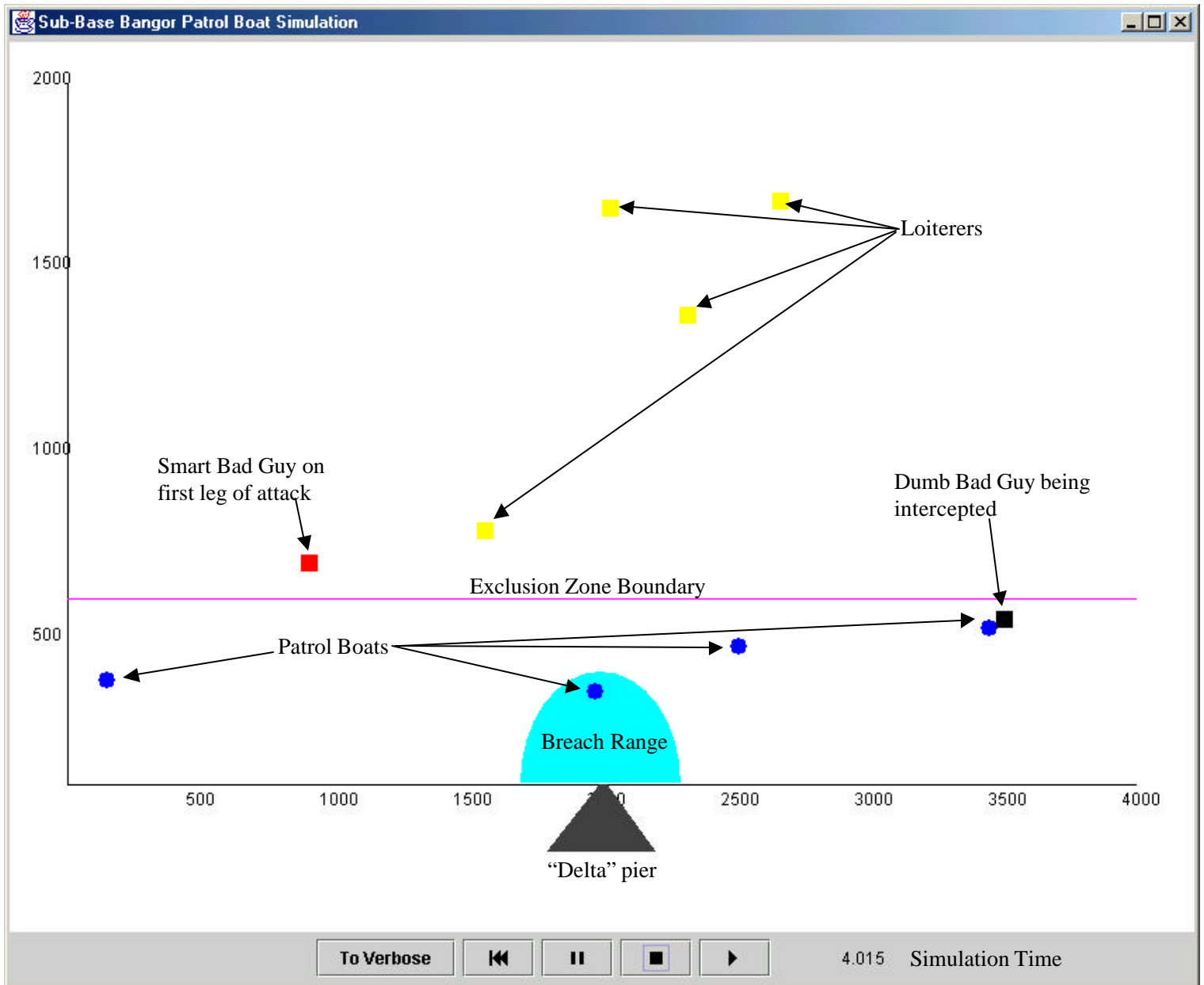


Figure 12. Graphical Display of Simulation with Notation Added

The River panel is drawn or "painted" through the addition of another "loosely couple component," the Animation class developed by Buss. Simulation animation can be turned on or off dependent upon what the model is being used for. For production analytical runs, the graphics are best turned off to allow rapid replication. For run-by-run intuitive analysis, the graphics are extremely valuable. Even before statistics are gathered, visual observation of a given Patrol Boat configuration against a specific attack

can give broad insights into the relative worth of different tactics. Beyond that, it is reassuring for analyst and decision-maker alike to have a way of "seeing" simulation analysis results other than some output file filled with numbers.

IV. MODEL RUNS AND ANALYSIS

A. EXPERIMENTAL DESIGN

1. Fundamentals: Repeatability and Replication

Before statistics are defined or gathered, two critical design elements, specific to simulation, must be established: replicability and repeatability. An experiment is *replicable* if a number of observations can be taken under a given set of conditions. It is *repeatable* if the conditions under which the data are collected can be reproduced. This Force Protection simulation experiment is replicable at two levels: within each *design point*, and across all design points of the experiment. Each design point of the experiment runs a specific Patrol Boat configuration against the same range of terrorist attacks. Each design point is run against each type of attack 500 times. Each of the 500 individual model runs is a *replication* of the simulation *within* a design point. This is the first level of replicability within the experiment. In order to make a true comparison among Patrol Boat configurations alternatives, the experiment must be repeatable. That means each configuration or design point must be faced with the very same situations as all other design points. For example, the 40th replication of design point eight against attack number three must present the same threat profile—the same number of threats in the same places at the same times—as the 40th replication of design points one through seven against attack number three. This is repeatability. After the experiment is repeated at each design point against all attacks, the results for a given attack *across* all design points is a single replication of the experiment. This is the second level of replicability within the experiment. A diagram may help to show what is meant by replication and repeatability in the context of the kind of experimental design employed to analyze the patrol boat problem.

Patrol Boat Parameters (inputs)					Outputs for model run number						
	# Boats	Patrol speed	Intercept speed	Patrol pattern	1	2	3	.	.	.	n
Setup one (Design point 1)	2	2	15	1	#	#	#	#	#	#	#
Setup two (Design point 2)	4	2	15	2	#	#	#	#	#	#	#
Setup three (Design point 3)	2	5	15	2	#	#	#	#	#	#	#
Setup four (Design point 4)	4	5	15	1	#	#	#	#	#	#	#
Setup five (Design point 5)	2	2	25	2	#	#	#	#	#	#	#
Setup six (Design point 6)	4	2	25	1	#	#	#	#	#	#	#
Setup seven (Design point 7)	2	5	25	1	#	#	#	#	#	#	#
Setup eight (Design point 8)	4	5	25	2	#	#	#	#	#	#	#

Entire column represents one replication of the **experiment**

Each number is the result of a single model run—one “replication” within one **design point**

Every output number within each column is a response to the same traffic conditions (**Repeatability**). “#” represents statistical output, which means more than one number. In fact, seven statistics are gathered for each model run.

Table 1. Representation of Repeatability and Replication within an Experiment

Law and Kelton say that, in general, replicability is achieved when:

Each run uses separate sets of different random numbers.

Each run uses the same initial conditions.

Each run resets the statistical counters.[Ref. 10; p. 212]

The assurance of separate sets of random numbers for each model run within a configuration run is achieved as described below, using random number generation capacities within Simkit. The second two criteria for replicability are met through "reset" methods within all classes that contain state variables and all statistics gathering classes.

Simkit has a *Schedule.reset()* method which, when invoked, calls all other reset methods within the simulation thereby re-establishing initial conditions. The simulation program itself contains a "Resetter" method that resets all statistical counters. Replication is particularly important in this simulation because it is a "terminating simulation," which means it ends upon a specific event, rather than running long enough to reach some sort of steady-state condition. [Ref. 10; p. 502] In particular, each simulation run terminates at a set time, after a terrorist attack has been attempted—successfully or not. In a terminating simulation, the only way to arrive at good statistics (statistics with reasonable confidence intervals) is to take many observations, which is to perform many model runs.

The assurance of repeatability from configuration run to configuration run is achieved, in simulation, through the proper use of random numbers:

The basic idea is that we should compare the alternative configurations "under similar experimental conditions" so that we can be more confident than any observed differences in performance are due to differences in the system configuration rather than to fluctuations of the "experimental conditions." In simulation, these "experimental conditions" are the generated random variates that are used to drive the models through simulated time. In queueing simulations, for instance, these would include interarrival times and service requirements of customers. [Ref. 10; p. 583]

The Patrol Boat simulation is a queueing system with randomly generated interarrival times, initial positions and initial speeds for Loiterers and Passersby. The goal is to ensure that each Patrol Boat patrolling configuration is tested under the very same traffic conditions. That goal is achieved through the use of Simkit's RandomVariate class that allows for specified seeded random number generation. Every configuration run, which consists of many model runs or "observations" at a specific input setup or design point, iterates through the very same sequence of interarrival times and random location and speed assignments for all Loiterer and Passerby objects. (In Table 2 below, we specify 500 model runs per configuration run.) Any differences in statistical output for a given kind of attack scenario, are the direct result of changes made to the Patrol Boat configuration. This allows for meaningful analysis.

2. Statistics

The specific goal of this Force Protection experiment—that has simulation results as its data—is to find Patrol Boat tactics that are most effective against a range of possible terrorist attacks under similar (actually identical) traffic conditions. Therefore, the experiment must first collect statistics that indicate Patrol Boat effectiveness. Each experimental design point pits a specific Patrol Boat configuration against a specific attack configuration and collects the following statistics: the number of threats intercepted, the average range of a threat from the HVU at intercept, the average number of available PatrolBoats (read servers), the number of threats that could not be intercepted because of infeasibilities, the number of "terrors" or successful terrorist attacks on the HVU, the number of threats inside the "breach range" from the HVU, and the average number of threats in the threat queue. All statistics are gathered by SimpleStats—the statistics gathering object in Simkit—instances that record FirePropertyChanges fired through the course of a simulation run.

a. Number Intercepted

The number of threats intercepted in a simulation run is a simple count (a "Tally" statistic) of how many threat contacts were intercepted. Because of the repeatability achieved through the RandomVariate random number generation, this provides a baseline measure of relative performance from one Patrol Boat configuration to another.

b. Range at Intercept

The average range of threats upon their interception refines the measure of relative performance by providing a more qualitative measure. All else being equal, a Patrol Boat configuration that intercepts at a greater range from the HVU is preferable.

c. Number of Available Patrol Boats

The number of available Patrol Boats is a time-varying statistic that measures Patrol Boat utilization, defined in the dynamic service model as the proportion of time a given configuration's Patrol Boats are actually moving to intercept threats. This statistic speaks to efficiency, giving the capacity to get a sense of the point of diminishing returns in the number of Patrol Boats necessary adequately to defend the Delta Pier (under the attack assumptions of the model).

d. Number of Leakers

The number of leakers is a tally statistic that records the number of threats that cannot be intercepted. The three kinds of intercept infeasibility mentioned with the derivation of intercept formulae in Chapter Two are tracked individually, but only the total number is recorded. Additionally, it is recognized that certain high speed randomly moving contacts will be "leakers" more than once in a model run, so a given leaker is only counted once. That is, the "leaker" statistic is actually a "unique leaker" statistic.

e. Number of Terrors

As the number of threats intercepted is the coarse estimate of how well a given Patrol Boat configuration does against a given attack, the number of "terrors" is the coarse estimate of how poorly it does—or, conversely, how well a terrorist attack configuration does. For each simulation run, the number of terrors will be less than or equal to the total number of terrorists in that run's attack.

f. Number of Breaches

The number of breaches is very similar to the number of leakers. It is a statistic that tallies the number of threats that make it inside the stand-off range from the HVU. It refines the "terrors" statistic by capturing the qualitative effectiveness of a Patrol Boat configuration in actually preventing a terrorist attack. As mentioned above, it may be a bit naïve to claim victory in the case where a terrorist object is successfully intercepted, but only a few yards from the HVU, at which range it may well have been able to achieve its goal. In any case, it is desirable that a Patrol Boat configuration hold as many contacts as possible outside the established stand-off range.

g. Average Number in Queue

The average number of threats in the threat queue—"in line" to be intercepted—is a time-varying statistic that gives a sense of the adequacy of Patrol Boat capabilities. A higher average number in the threat queue indicates an inability of Patrol Boats to respond to all threats, suggesting vulnerability.

3. Run Setup: Fractional Factorial Design

This thesis does not analyze every possible combination of Patrol Boat parameters for every conceivable terrorist attack scenario, but it does provide an analysis of defense against some feasible attacks while varying specific Patrol Boat parameters of interest to

the CSG-9 security officer. It is hoped that the reader will gain insights and, more importantly, see a demonstration of a basic methodology that could be used and/or expanded for future model analyses.

Once output statistics have been defined, it is time to specify the simulation's experimental design. The simulation runs are set up so that enough observations can be gathered to provide good estimates, while at the same time providing insights into the effects of changing a number of Patrol Boat input factors. Factorial design provides an excellent vehicle for the kind of exploratory analysis that this model is meant to achieve. Specifically, two-level factorial designs allow for the analysis of two levels of a number of input factors simultaneously. Though this design approach does not allow all possible levels of all possible factors to be analyzed, it does provide a very powerful means of identifying the factors whose "main effects"—independent contributions—dominate outputs. As Box, Hunter and Hunter point out, these kinds of designs are exceptionally efficient and very practical:

They require relatively few runs per factor studied; and though they are unable to explore fully a wide region in the factor space, they can indicate major trends and so determine a promising direction for further experimentation.[Ref. 24; p. 306]

Because so many parameters can be varied in the Force Protection model, a two-level factorial design is the ideal vehicle for showing its utility in providing insights into specific scenarios. In particular a two-level *fractional factorial* design will be used to allow for analysis of four different Patrol Boat factors with just eight design points. As two-level factorial design limits the size of the factor space that can be explored, fractional factorial design limits the number of higher order interactions that can be explored. This means that the design assumes a factor's effect in acting by itself—its main effect—dominates its effect when acting together with one or more other factors. This assumption can be justified by the fact that as the number of variables or factors (k) introduced to a model increases:

at some point higher order interactions tend to become negligible and can properly be disregarded . . . there tends to be redundancy in a 2^k design if k is not small—redundancy in terms of excess number of interactions that can be estimated and sometimes in an excess of number of variables that

are studied. Fractional factorial designs exploit this redundancy. [Ref. 24; p. 374-75]

The practical justification for using a fractional factorial design in this analysis is that the analysis is exploratory and more concerned with trends than point estimates. Additionally, the fractional factorial design in no way precludes analysis of higher order interactions, but rather can give insights to where they may exist, so that more focused analysis can be performed.

Two levels (a high and a low) of four different Patrol Boat factors will be used against eight different terrorist attack configurations. The Patrol Boat factors to be varied are: the number of Patrol Boats, with levels of 2 and 4; patrolling speed, with levels of 2 and 5 knots; intercept speed, with levels of 15 and 25 knots; and patrol pattern, with "levels" of barrier (1) and "bow-tie" or race-track(2). The eight different terrorist attack configurations are comprised of attacks by either one or two terrorists: 1) one Dumb Bad Guy, 2) one Smart Bad Guy, 3) two Dumb Bad Guys in a *coordinated* attack, 4) two Dumb Bad Guys in a *synchronized* attack, 5) two Smart Bad Guys in a *coordinated* attack, 6) two Smart Bad Guys in a *synchronized* attack, 7) one Smart Bad Guy and one Dumb Bad Guy in a *coordinated* attack, 8) one Smart Bad Guy and one Dumb Bad Guy in a *synchronized* attack. A coordinated attack is defined as one where both terrorists start from the same point at the same time. A synchronized attack is defined as an attack where terrorists start from opposite ends of the river at the same time. The following diagrams (Figures 13 through 16) are representative of different set-ups, though not all-inclusive. One point of interest is that patrolling stations for patrol boats are established so there is full "horizontal" coverage of the area in front of the pier regardless of the number of patrol boats.

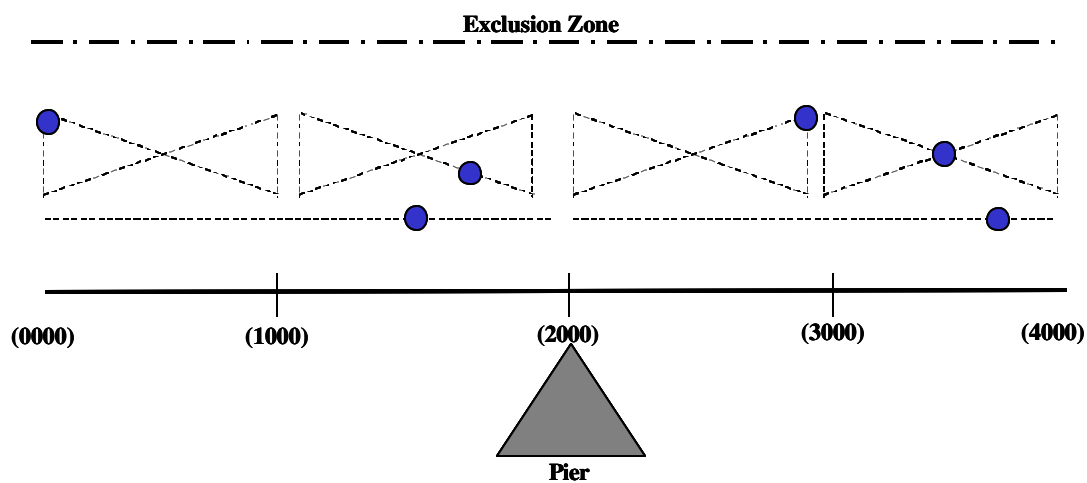


Figure 13. Patrol Patterns: 4 PB's with bow-tie pattern and 2 PB's with barrier pattern shown

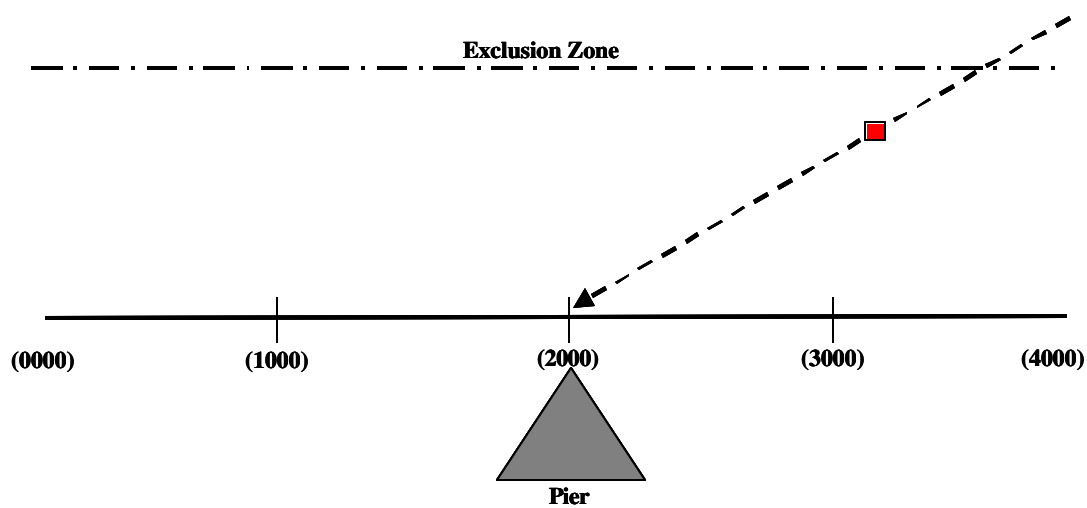


Figure 14. Attack by One Dumb Terrorist

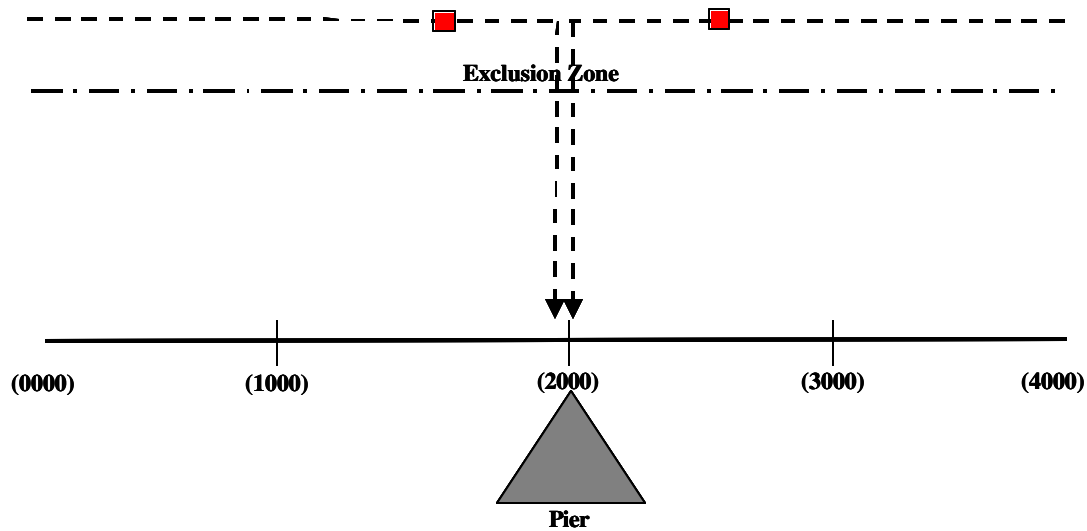


Figure 15. Synchronized Attack by Two Smart Terrorists

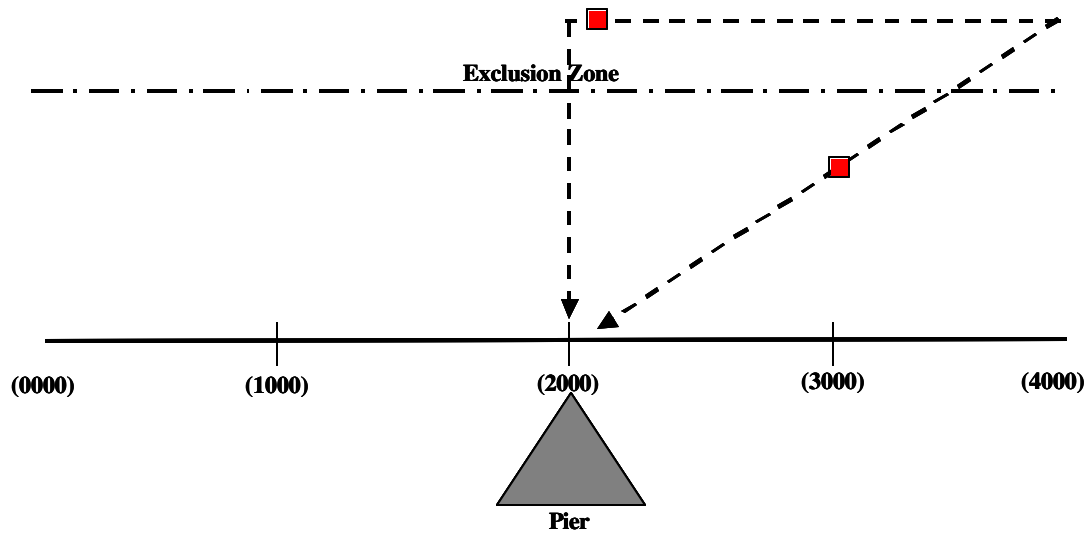


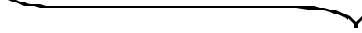
Figure 16. Coordinated Attack by One Smart and One Dumb Terrorist

Patrol Boat values are notional, since this is a non-classified thesis, but are reasonable small boat values and tactics. Terrorist attack scenarios are designed to address CSG-9 concerns, and are based upon the findings of the *Cole* report cited in Chapter One, but with the addition of another craft—which could be extended to more

than two if desired—to reflect the possibility that more than one boat may be used as evidenced by the 11 September coordinated air attacks.

With all the pieces in place, the application of the fractional factorial experimental design to the Patrol Boat factors and terrorist attacks of interest results in an 8 x 8 matrix (Table 2).

Patrol Boat Parameters (inputs)					Terrorist Attack Configuration							
	# Boats	Patrol speed	Intercept speed	Patrol pattern	1	2	3	4	5	6	7	8
Setup one (Design point 1)	2	2	15	1	500 runs	500 runs	500 runs	500 runs	500 runs	500 runs	500 runs	500 runs
Setup two (Design point 2)	4	2	15	2
Setup three (Design point 3)	2	5	15	2
Setup four (Design point 4)	4	5	15	1
Setup five (Design point 5)	2	2	25	2
Setup six (Design point 6)	4	2	25	1
Setup seven (Design point 7)	2	5	25	1
Setup eight (Design point 8)	4	5	25	2	500 runs



Shaded area = output matrix

Table 2. Fractional Factorial Design of Complete Experiment

Here the column headings are attack type rather than run number. Each output row of the earlier matrix represents one output element of this matrix, which is to say each element of this experiment matrix involves 500 runs of the model with a given Patrol Boat configuration against a given attack type. Therefore, the entire experiment consists of 8 design points, yielding 64 multivariate data points or design observations, generated from 32,000 (64*500) individual observations or model runs. Each 500 run observation generates the statistics described above for use in analysis of Patrol Boat performance.

B. RESULTS AND ANALYSIS

Raw data from the model runs were collected in Excel spreadsheets. There is one spreadsheet per design point. Here (Table 3), the raw data sheet for design point one is shown; all sheets are displayed in Appendix B.

		breach- count	leakers- count	numberAv ailablePBs mean	numberInt erceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp1.1	mean	2.93988	13.4509	1.830053	0.004025	30.88577	957.3882	0.152305
	sd	1.768872	3.781042	0.051879	0.046242	5.095964	86.19461	0.359676
	var	3.128908	14.29628	0.002691	0.002138	25.96885	7429.511	0.129367
dp1.2	mean	3.661323	14.48297	1.838598	0.001172	30.18437	954.2404	0.861723
	sd	1.632848	3.811472	0.050074	0.002704	5.220641	81.17603	0.345536
	var	2.666192	14.52732	0.002507	7.31E-06	27.2551	6589.548	0.119395
dp1.3	mean	3.881764	14.41683	1.835858	0.00449	31.08617	954.6288	0.991984
	sd	1.68028	3.702083	0.049546	0.049565	5.177821	83.28889	0.533925
	var	2.823341	13.70542	0.002455	0.002457	26.80983	6937.039	0.285076
dp1.4	mean	3.336673	14.12024	1.833563	0.00309	31.42285	960.5784	0.637275
	sd	1.758193	3.925712	0.050049	0.033851	5.381767	81.91754	0.61335
	var	3.091243	15.41122	0.002505	0.001146	28.96341	6710.484	0.376198
dp1.5	mean	4.643287	15.47295	1.837511	0.009546	30.26453	956.5379	1.843687
	sd	1.658921	3.904067	0.048999	0.08931	5.189014	90.34237	0.390159
	var	2.752018	15.24174	0.002401	0.007976	26.92587	8161.745	0.152224
dp1.6	mean	4.521042	15.26052	1.838607	0.002538	30.30661	953.5084	1.851703
	sd	1.648005	3.957133	0.048999	0.020218	5.232753	85.98333	0.372299
	var	2.715922	15.6589	0.002401	0.000409	27.3817	7393.134	0.138607
dp1.7	mean	3.897796	14.52705	1.838172	0.017015	30.94188	953.5896	1.114228
	sd	1.684913	3.894797	0.047951	0.120719	5.283397	83.8448	0.430953
	var	2.838931	15.16945	0.002299	0.014573	27.91429	7029.95	0.185721
dp1.8	mean	3.857715	14.58918	1.836917	0.018639	30.998	956.0409	1.114228
	sd	1.711582	4.034684	0.049164	0.144296	5.309497	82.83713	0.466743
	var	2.929514	16.27868	0.002417	0.020821	28.19076	6861.991	0.217849
	$\bar{Y}_{1.}$	3.842435	14.54008	1.83616	0.007564	30.76127	955.8141	0.598823
	$S^2_{1.}$	2.868259	15.03612	0.00246	0.006191	27.42623	7139.175	0.07346
	Var(means)	0.315703	0.396126	8.93E-06	4.63E-05	0.204812	5.699677	0.081763
	V_1	3.183961	15.43225	0.002469	0.006237	27.63104	7144.875	0.155223

Table 3. Output Data for Design Point One

The table shows the mean, standard deviation, and sample variance for each statistic for each Patrol Boat configuration against each of the eight types of terrorist attacks. The labeling in the first column of the table corresponds to the elements of the 8x8 matrix shown in Table 2. For example, the numbers following the label "dp1.1" are the means, standard deviations and sample variances for all statistics collected from the 500 model runs that pit the first Patrol Boat configuration (two patrol boats in barrier patterns, with patrolling speeds of 2 kts and intercepting speeds of 15 knots) against the first type of terrorist attack (one Dumb Bad Guy). These values correspond to the upper left hand element (element 1,1) of the output matrix of Table 2.

The last four rows of data in Table 3 are statistics for the Design Point as a whole. They are the statistics that provide the measure of how well the Patrol Boat configuration of Design Point One performed across the full spectrum of postulated attacks. These design point statistics are used to generate the experiment's "metamodel" through regression analysis. Each design point's statistics are the result of its 8 * 500 simulation runs. Those statistics are, in other words, the collective result due to the changes in input variables (Patrol Boat parameters) that define each design point. In the case of the "breach-count" statistic, for example, \bar{Y}_1 is the average of all the mean values in the "breach-count" column (i.e. the average of 2.93988, 3.661323 . . . 3.857715), $S^2_{i.}$ is the average of all the sample variance values, $\text{Var}(\text{means})$ is the sample variance of the sample means, and V_1 is the total variance for the "breach-count" statistic for Design Point One— $V_1 = S^2_{i.} + \text{Var}(\text{means})$. The same statistics are gathered at each design point.

These design point summary statistics are used to implement the "Robust design philosophy" that considers both mean and total variance output values at each design point to select the best configuration option available.[Ref. 25] The idea is that the mean output at a design point does not tell the whole story of a configuration's performance. A design point that has a very desirable mean value might have much higher variance around that mean than another point with a slightly less desirable mean. For example, consider a design point with a "breaches" mean of 3 and variance of 6 compared to another with a mean of 4 and variance of 2. Assume the target mean value is 3 breaches. A comparison of the mean values would lead to the former point's selection, but the

robust design selection would be the latter, because it provides a close to optimal outcome with more certainty. The total variance term in the Robust design scheme is a combination of variance inherent to the data within a design point ($S^2_{i.}$) and the variance in the (mean) statistical output "across the design point" ($V(\text{mean})$). This means the robust design allows for analysis both of output level and variance about an output level, so output selection is based upon more than a point estimate.[Ref. 25 ; p. 289-90]

The bottom line is that analysis of the eight Patrol Boat configurations simulated is performed by regressing the design point means and variances against the Patrol Boat input variables. For a brief discussion of linear regression see Appendix C. Regressions were performed for four output statistics: terrors, leakers, breachers, and range at intercept.

1. "Terrors" Regression Models

The data for the models are:

Terrors							
	numPBs	patSpd	intSpd	patPat		$\bar{Y}_{i.}$	V_i
designPoint1	2	2	15	1		0.598823	0.155223
designPoint2	4	2	15	2		0.411323	0.188209
designPoint3	2	5	15	2		0.586172	0.162351
designPoint4	4	5	15	1		0.411072	0.186097
designPoint5	2	2	25	2		0.506388	0.161764
designPoint6	4	2	25	1		0.306613	0.157854
designPoint7	2	5	25	1		0.490606	0.170289
designPoint8	4	5	25	2		0.374248	0.183041

Table 4. Data for Regression Model for "Terrors" Statistic

For the general linear regression model, $Y = \beta X$, the "X" input variables, X_1 through X_4 , are numPBs, patSpd, intSpd and patPat, or number of Patrol Boats, patrolling speed, intercept speed and patrol pattern. Those inputs are used to generate one regression model for the mean and another for total variance. In all regression models, terms with p-values of .05 or less are considered statistically significant. The regression model for the mean "terrors" produced the following results, with an R^2 value of .974 and an adjusted R^2 of .939, which indicates the model accounts for 94% of the variance in the "terrors" mean realized over the eight design points is accounted for by the model.

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>lower 95.0%</i>	<i>upper 95.0%</i>
Intercept	0.460655686	0.008994571	51.21486	1.64E-05	0.4320309	0.48928	0.432031	0.48928
numPBs	-0.084841558	0.008994571	-9.43253	0.002525	-0.1134663	-0.056217	-0.113466	-0.056217
patSpd	0.004869113	0.008994571	0.541339	0.625889	-0.0237557	0.033494	-0.023756	0.033494
intSpd	-0.041191759	0.008994571	-4.579625	0.019545	-0.0698165	-0.012567	-0.069817	-0.012567
patPat	0.008877129	0.008994571	0.986943	0.396437	-0.0197476	0.037502	-0.019748	0.037502

Table 5. Regression Output for "Terrors" Mean

The statistically significant variables are the number of Patrol Boats and Intercept speed. For the range of attacks the probability a terrorist will succeed in an attack varies from a low of about .3 at design point six to a high of about .6 at design point one—a lower value is better. Hence there is a one hundred percent improvement realized by varying the Patrol Boat configuration. The magnitude of the coefficient values for the significant input variables indicates how one “unit” of change in the inputs affects the output. Because the low and high variable levels were actually run set at +1 and –1 for uniformity’s sake, the total unit change for each input variable is two units. This means that one Patrol Boat unit is one Patrol Boat (since its actual range is also two, 2-4); and one intercept speed unit is five knots (since its range is actually ten knots). Therefore, for each additional patrol boat, an 8.48% improvement (or decrease in P(success) for the terrorists) is achieved; and for each five knot increase in intercept speed a 4.14% improvement is achieved. The coefficients are negative because the input variable values are inversely related to the output value: increasing Patrol Boats and intercept speed decreases a terrorist attack’s likelihood of success.

The mean regression results can be obtained or confirmed graphically, in a way which is perhaps more intuitive. The following demonstrates a very powerful analytical technique: looking at the data! A Graph of the terrors results for each design point versus each attack type clearly shows that design point six consistently yields the best result:

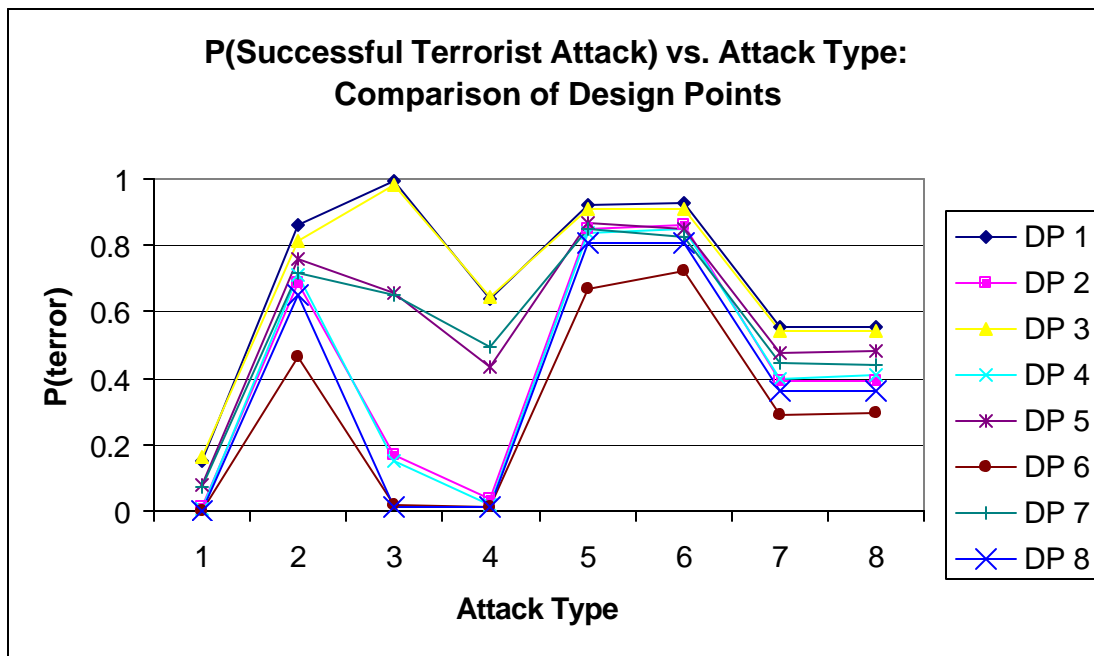


Figure 17. Graphical Confirmation of Mean Regression Result

Figure 17 provides a summary of the relative performance of each design point across the range of proposed attacks. The graph is generated from the raw data output of all design points (seen Table 3 for design point one, and Appendix B for all other design points).

Further graphical displays confirm the significance of varying the levels of the two factors (number of Patrol Boats and Intercept Speed) identified as statistically significant through the regression output, and the concomitant insignificance of changing the levels of the remaining two factors (Patrolling Speed and Patrol Pattern).

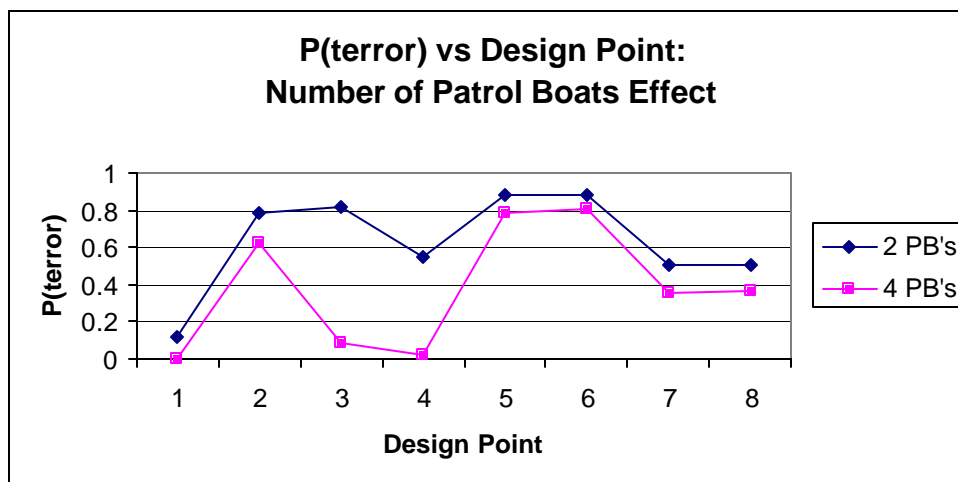


Figure 18. Graphic Showing the Advantage of Four Patrol Boats Over Two

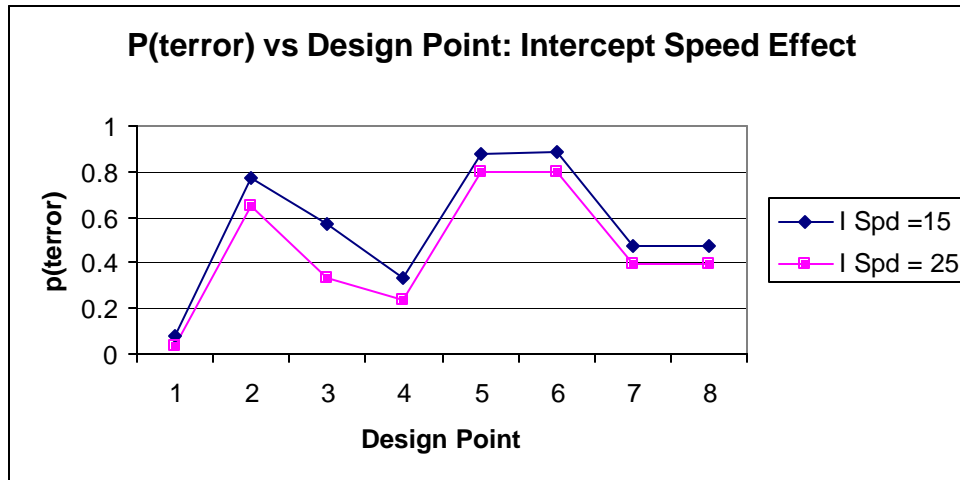


Figure 19. Graphic Showing the Advantage of Higher Intercept Speed

The increase in performance—decrease in $P(\text{terror})$ —is, perhaps, more obvious in Figure 18 than Figure 19, but both graphs confirm that changes in these two factors produce clearly better outcomes. The pictures tell in an instant what the regression model indicates through its t -statistics/ p -values: it is always better to have more Patrol Boats, and always better to intercept at a higher speed.

The graphs of the changes in $P(\text{terror})$ for changes in Patrolling Speed and Patrol Pattern show that they have no significant effect:

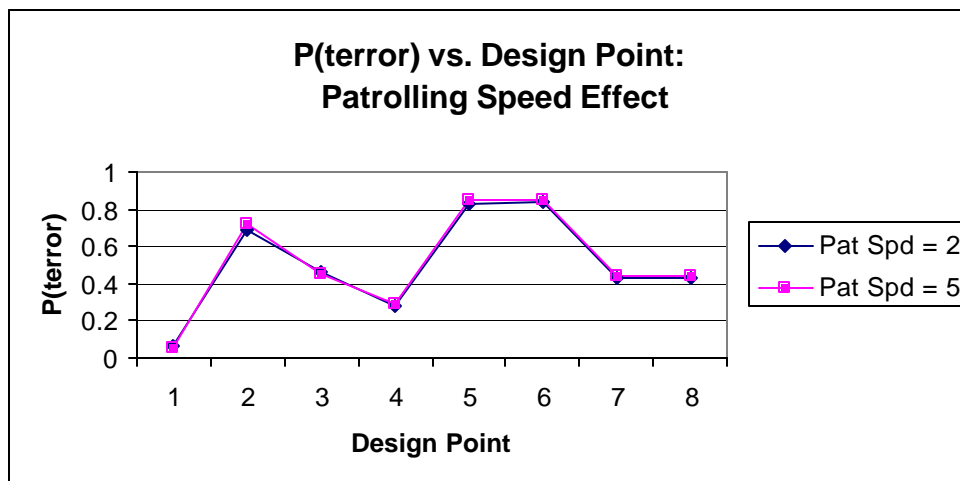


Figure 20. Changes in Patrolling Speed Have No Effect Upon Output

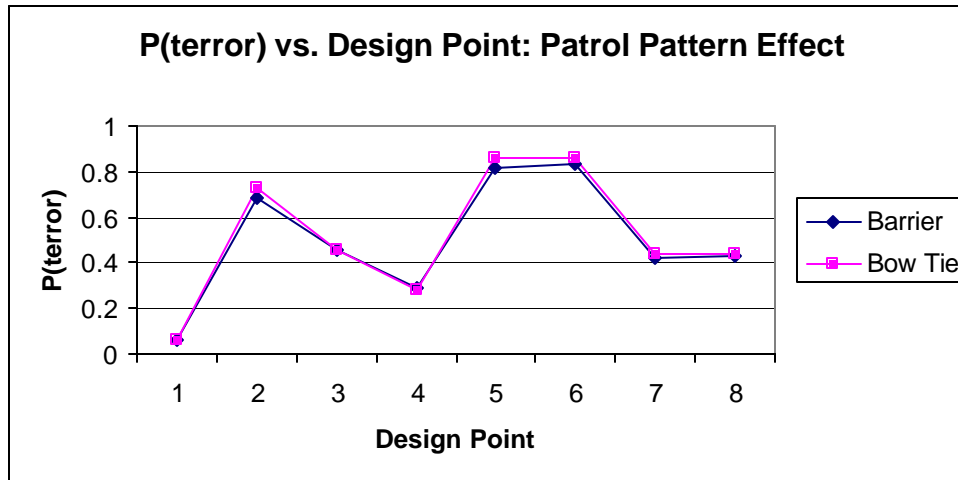


Figure 21. Changes in the Patrol Pattern Have No Effect Upon Output

Once again these two graphs (Figures 20 and 21) capture the meaning of the regression model's t-statistics/p-values for the factors depicted.

One final graphical display, a factor interaction plot, confirms the inclusion of only main effects in the regression model, by showing there are no interactions between the statistically significant factors. This simple but powerful plot also provides a sense of the relative worth of the two significant factors. The assumption that main effects dominate, which was made for efficiency's sake in approaching this exploratory analysis, is shown to have been a reasonable one:

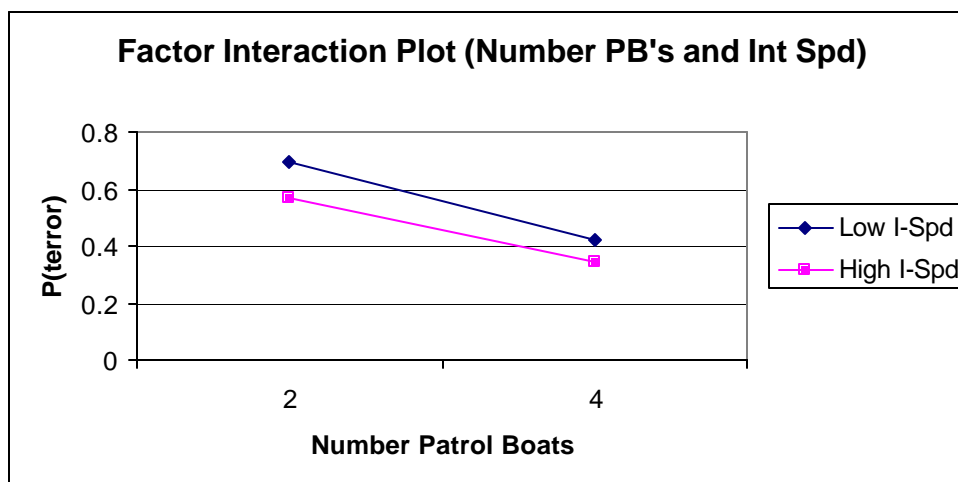


Figure 22. No Interaction Between Significant Factors

The fact that the two lines are virtually parallel indicates that the benefit of increasing intercept speed does not depend upon the number of Patrol Boats and vice versa.

Another way of looking at this is that, for each line in the plot, the slope change defined by its end points is dependent only upon the number of Patrol Boats. Hence the two factors can be considered independently, which is very important in the analysis of the problem, since the opposite case—a situation where an interaction were present—would mean no conclusions could be drawn about the importance of *either* factor without considering *both* factors.

One other insight gained from the interaction plot concerns the relative effect of the two significant factors upon the P(terror) measure of performance. The change in P(terror) due to the change in the number of Patrol Boats can be measured horizontally *across* the plot from one end of each line to the other; and the change due to intercept speed vertically, from the corresponding end points of the separate lines:

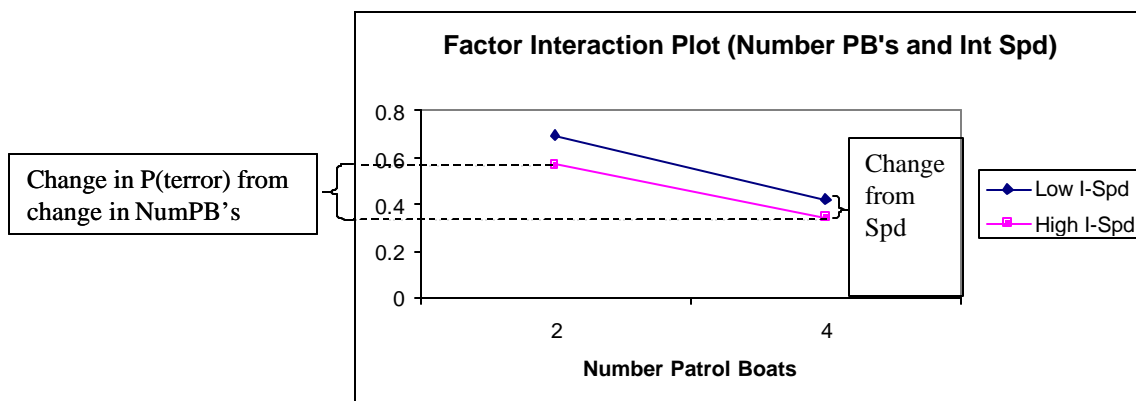


Figure 23. Relative Effects of Significant Factors Upon P(terror) Outcome

This analytical insight allows a decision maker to determine which of the two significant factors in the model yields the greater benefit, should a choice between the two be necessary because of some exterior constraint—such as budgeting. In this case, the effect of increasing Intercept Speed is greater than that of increasing the number of Patrol Boats, which is generally consistent with the other measures of performance.

Similar plots for all statistics analyzed are included in Appendix D. In all cases the mean regression models are confirmed—including the omission of interaction terms.

The results for the total variance model had an R^2 of .683, with an adjusted R^2 of .260:

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>Lower 95.0%</i>	<i>Upper 95.0%</i>
Intercept	0.170603814	0.004064567	41.97342	2.98E-05	0.1576685	0.183539	0.157669	0.183539
numPBs	0.008196867	0.004064567	2.016664	0.137097	-0.0047384	0.021132	-0.004738	0.021132
patSpd	0.0048411	0.004064567	1.191049	0.319281	-0.0080942	0.017776	-0.008094	0.017776
intSpd	-0.002366582	0.004064567	-0.582247	0.601271	-0.0153019	0.010569	-0.015302	0.010569
patPat	0.003237603	0.004064567	0.796543	0.483928	-0.0096977	0.016173	-0.009698	0.016173

Table 6. Regression Output for "Terrors" Total Variance

The low R^2 value indicates that there is no significant variance in the terror statistic from design point to design point. The variance in the terror statistic within the metamodel exists primarily within the individual design points rather than across them. This means that metamodel variance can be assumed constant, and that the mean value regression alone is the best indicator of the optimal design point or Patrol Boat configuration. Hence, for this measure of effectiveness, design point six, which has the lowest mean value, is the optimal configuration.

2. "Leakers" Regression Models

The data for the models are:

Leakers						
	numPBs	patSpd	intSpd	patPat	Ybar _i	V _i
designPoint1	2	2	15	1	14.54008	15.4322502
designPoint2	4	2	15	2	8.473697	8.54557976
designPoint3	2	5	15	2	14.88477	14.9307669
designPoint4	4	5	15	1	6.933116	7.04577238
designPoint5	2	2	25	2	9.19489	9.56027988
designPoint6	4	2	25	1	2.825902	2.80819265
designPoint7	2	5	25	1	8.372996	8.46694949
designPoint8	4	5	25	2	3.480711	3.24499689

Table 7. Data for Regression Model for "Leakers" Statistic

The regression model for the mean number of leakers across the design points has an R^2 value of .992 and an adjust R^2 value of .981, indicating the model explains virtually all the variance in the mean output.

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>Lower 95.0%</i>	<i>Upper 95.0%</i>
Intercept	8.58827	0.214392521	40.05863	3.42E-05	7.905976965	9.270564	7.90597697	9.270564
numPBs	-3.159914	0.214392521	-14.73892	0.000678	-3.8422069	-2.47762	-3.8422069	-2.47762
patSpd	-0.170372	0.214392521	-0.794673	0.484865	-0.85266532	0.511921	-0.85266532	0.511921
intSpd	-2.619646	0.214392521	-12.21892	0.00118	-3.30193887	-1.937352	-3.30193887	-1.937352
patPat	0.420247	0.214392521	1.960174	0.144828	-0.26204658	1.10254	-0.26204658	1.10254

Table 8. Regression Output for "Leakers" Mean

The same two input variables (number of Patrol Boats and intercept speed) are again statistically significant, and negative to show that with their increases, the number of leakers goes down. With each additional Patrol Boat, the number of leakers is projected to decrease by 3.16; and with each 5 knot increase in intercept speed it is projected to decrease by 2.62.

The regression model for leaker variance across all design points also has an R^2 value of .992 and an adjust R^2 value of .981

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>Lower 95.0%</i>	<i>Upper 95.0%</i>
Intercept	8.754349	0.227279205	38.51804	3.85E-05	8.031043975	9.477653	8.03104397	9.477653
numPBs	-3.343213	0.227279205	-14.70972	0.000682	-4.06651764	-2.619909	-4.06651764	-2.619909
patSpd	-0.332227	0.227279205	-1.461758	0.239967	-1.05553165	0.391077	-1.05553165	0.391077
intSpd	-2.734244	0.227279205	-12.03033	0.001236	-3.45754834	-2.010939	-3.45754834	-2.010939
patPat	0.316057	0.227279205	1.390613	0.258545	-0.40724721	1.039362	-0.40724721	1.039362

Table 9. Regression Output for "Leakers" Total Variation

The statistically significant variables within the variance model are the same as in the mean model, indicating that they drive metamodel variance as well as mean. In this case the variance from one design point to another cannot be assumed constant, and must be considered in selecting most "robust" design point for this measure of effectiveness. A comparison of the regression models as well as of the data themselves shows that leaker variance tracks with its mean: as the mean goes up, so does variance, and vice versa. This means there is no concern about choosing a "false optimal" design point by considering the mean alone. This result strongly confirms the selection of the design point with best mean value, since that design point also has the lowest variance about its mean. Again the best design point is Patrol Boat configuration number six.

3. "Breaches" Regression Models

The data for the models are:

Breaches						
	numPBs	patSpd	intSpd	patPat	$\bar{Y}_{i\cdot}$	V_i
designPoint1	2	2	15	1	3.842435	3.183961
designPoint2	4	2	15	2	2.28482	2.161019
designPoint3	2	5	15	2	3.842184	3.266478
designPoint4	4	5	15	1	2.330912	2.105665
designPoint5	2	2	25	2	2.318136	1.848684
designPoint6	4	2	25	1	1.406563	1.264091
designPoint7	2	5	25	1	2.550852	2.072597
designPoint8	4	5	25	2	1.438377	1.343602

Table 10. Data for Regression Models for "Breaches" Statistic

The regression model for the mean number of breaches per design point has an R^2 value of .976 and an adjust R^2 value of .944.

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>lower 95.0%</i>	<i>upper 95.0%</i>
Intercept	2.501785	0.077842076	32.13924	6.62E-05	2.25405636	2.749513	2.254056	2.749513
numPBs	-0.636617	0.077842076	-8.178315	0.003825	-0.88434544	-0.388889	-0.884345	-0.388889
patSpd	0.038796	0.077842076	0.498398	0.652452	-0.20893212	0.286525	-0.208932	0.286525
intSpd	-0.573303	0.077842076	-7.364948	0.005174	-0.82103131	-0.325574	-0.821031	-0.325574
patPat	-0.030906	0.077842076	-0.397029	0.717905	-0.27863402	0.216823	-0.278634	0.216823

Table 11. Regression Output for “Breaches” Mean

Here, the model coefficients, that indicate the per unit change in the number of breaches, are smaller than the coefficients for the number of leakers, but that difference in magnitude corresponds with the difference in magnitude between the average number of leakers and breaches. The number of Patrol Boats is still has the dominant effect upon the outcome.

The regression model for the variation in the number of breaches across the design points has an R^2 value of .970 and an adjust R^2 value of .931.

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>lower 95.0%</i>	<i>upper 95.0%</i>
Intercept	2.155762	0.068965181	31.2587	7.19E-05	1.93628384	2.37524	1.936284	2.37524
numPBs	-0.437168	0.068965181	-6.338966	0.00794	-0.65664611	-0.21769	-0.656646	-0.21769
patSpd	0.041323	0.068965181	0.599191	0.591275	-0.17815485	0.260802	-0.178155	0.260802
intSpd	-0.523519	0.068965181	-7.591057	0.004743	-0.74299684	-0.30404	-0.742997	-0.30404
patPat	-0.000816	0.068965181	-0.011838	0.991298	-0.22029457	0.218662	-0.220295	0.218662

Table 12. Regression Output for “Breaches” Total Variance

The results for the “breaches” statistic of the metamodel are very similar to those of the “leakers.” Both regressions yield good models, and the model results track each other exactly. Again, the statistically significant variables in the Patrol Boat configuration are the number of boats and their intercept speeds; and again, because variance decreases along with the mean, the design point with the most favorable mean value also has the lowest variance about that value. The best point for the “breaches” statistic is design point six.

4. "Range at Intercept" Regression Models

The range at intercept statistic is the mean range of individual run mean ranges.

The data for the models are:

Intercept Range							
	numPBs	patSpd	intSpd	patPat		Ybar _i	V _i
designPoint1	2	2	15	1		955.8141	7144.87482
designPoint2	4	2	15	2		984.8698	6393.87433
designPoint3	2	5	15	2		960.3437	6685.79177
designPoint4	4	5	15	1		982.0539	6405.11773
designPoint5	2	2	25	2		984.2688	6780.72977
designPoint6	4	2	25	1		1010.97	6744.33899
designPoint7	2	5	25	1		998.3512	6490.6863
designPoint8	4	5	25	2		1015.697	7005.47552

Table 13. Data for Regression Models for "Range at Intercept" Statistic

The regression model for the mean range at intercept as a function of design point has an R^2 value of .976 and an adjusted R^2 value of .945.

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept	986.54606	1.791654631	550.6341	1.32E-08	980.844209	992.2479	980.844209	992.24791
numPBs	11.85161	1.791654631	6.614897	0.007035	6.14976022	17.55346	6.14976022	17.55346
patSpd	2.5654697	1.791654631	1.4319	0.247582	-3.1363803	8.26732	-3.1363803	8.2673197
intSpd	15.775674	1.791654631	8.805087	0.003086	10.0738244	21.47752	10.0738244	21.477524
patPat	-0.251179	1.791654631	-0.140194	0.89739	-5.953029	5.450671	-5.953029	5.450671

Table 14. Regression Output for "Range at Intercept" Mean

The statistically significant variables for the mean of this measure of performance are the same as for all others analyzed. In this one case the model coefficients are positive, since a higher range at intercept is preferable. With each additional Patrol Boat the intercept range increase by 11.8 yards, while with each 5 knot increase in intercept speed the range is move away from the HVU about 16 yards. Given that the average range at intercept is over 900 yards from the HVU at all design points, it may arguable as to whether or not changes of less than 20 yards are significant. In any case, a more desirable outcome is realized as a natural effect of improving other performance measures, whether this measure of performance is considered vital or not.

The regression model for the variation in the number of breaches across the design points has an R^2 value of .165 and an adjusted R^2 value of zero—the Excel output is negative, but that is nonsensical.

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>	<i>Lower 95.0%</i>	<i>Upper 95.0%</i>
Intercept	6706.3612	135.0019365	49.67604	1.8E-05	6276.72434	7135.998	6276.72434	7135.998
numPBs	-69.159513	135.0019365	-0.512285	0.643782	-498.79633	360.4773	-498.79633	360.4773
patSpd	-59.593324	135.0019365	-0.441426	0.688783	-489.23014	370.0435	-489.23014	370.04349
intSpd	48.946491	135.0019365	0.362561	0.74097	-380.69033	478.5833	-380.69033	478.58331
patPat	10.106695	135.0019365	0.074863	0.945036	-419.53012	439.7435	-419.53012	439.74351

Table 15. Regression Output for “Range at Intercept” Total Variance

As with the model for variance of the “errors” statistic, this model result indicates that variance in the statistic across the experimental design points can be assumed constant. This leaves the design point with the most desirable mean as the most desirable design point. Design point eight yielded the greatest mean range across the spectrum of proposed attacks, and is therefore the optimal design point for the “Range at Intercept” measure of performance.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. ANALYTICAL CONCLUSIONS

First, by way of a reality check, it should be recalled that the overarching goal of the Force Protection model's analysis is to provide *insights* into the effects of varying key Patrol Boat factors. It is very important to recognize that the goal of this or any simulation can never be to provide *the "right" answer* to the question of how to deploy assets. Regardless of the validity of any simulation, it can only produce results based upon how well input parameters represent reality. Certain simulation parameters are simply unknown or random, such as arrival rates of contacts into the river, or the time of a terrorist attack. These parameters must always remain estimates. Other tactical parameters, such as the number of Patrol Boats, Patrol Boat intercept and patrolling speeds, and patrolling patterns, are policy parameters and therefore are "known." Hence, while there can be a danger of looking to simulation outputs as some sort of Rosetta stone, which, if properly interpreted or analyzed will yield ground truth, it is also true that simulation results will provide very realistic estimates insofar as realistic values for any unknown parameters are used. The simulation tool becomes more valuable as the intelligence estimates of its parameter space improve.

That having been said, the results of the four statistical outcomes delineated above yield informative cumulative results and insights. The most apparent conclusion is Patrol Boat configuration effectiveness is primarily a function of the number of boats employed and their intercept speeds. In most cases, it seems that intercept speed is the most significant effect, though it is not exactly clear how the two variables' relative worths are precisely determined—the factor interaction plots are a good start. The interesting insight gained from this fundamental observation is that neither patrol speed nor the patrol pattern is significant. The practical value of this result is that the simplest pattern can be selected and the most efficient patrol speed used. Simplicity of pattern has the advantage of allowing a boat operator to expend most of his effort looking for threats rather than plotting his course. As for patrolling speed, it may well be that cutting a patrolling craft's speed by more than half will result in greater fuel efficiency and fewer

maintenance problems. In any case, the boat operators are not constrained to use a high patrolling speed.

As for the optimal Patrol Boat configuration, three of the four statistical outputs analyzed yielded design point six as the optimal configuration. Only the “Range at Intercept” statistic points to another (design point eight). The graphs of relative design point performance show that these two design points (six and eight) consistently yield similar results, which stands to reason. Since the number of Patrol Boats and the intercept speed are the only two significant input variables, design points six and eight are really the same design point, since they alone both hold those variables at the same level. Because neither the patrolling speed nor the patrolling pattern is significant, changes in their levels have no significant effect upon outcomes. Design points six and eight are the two configurations that hold both the number of patrol boats and their intercept speeds at their high levels. They are, therefore, *both* the optimal configuration, since they both have four Patrol Boats with intercept speeds of 15 knots.

B. RECOMMENDATIONS

The specific and limited recommendation derived from this exploratory analysis is to use larger numbers of patrol boats and higher intercept speeds. It is further recommended that the patrolling pattern and patrolling speed be determined based upon practical considerations specific to waterfront layout—for patrol pattern—and Patrol Boat characteristics—for speed. The more general recommendation is that the model be exercised further with more specific parameters determined by those tasked with the waterfront Force Protection mission. The model has given very reasonable results with notional data. It can be run locally with more sensitive data, and analyzed as demonstrated in Chapter Four, with the Excel Data Analysis Tool kit (or simply with “straight-stick” Excel graphics) generally available at government installations.

C. FOLLOW-ON WORK

Follow-on work is needed in two major areas: further software development, and more detailed analysis. The model extends Simkit's Movers and movement in at least two significant ways: it adds an intercept capacity, which is critical in many operational

applications, and effects a kind of agency within Movers through the TypedBasicMover extension. That notwithstanding, there is much more work to be done. The model was designed with loose coupling in mind, but there are many improvements to be made to allow more universal application. The ideal goal is to allow the end user to define both the playing field and players of the simulation. That ideal is currently only partially achieved. Additionally, follow-on software work should focus on generating graphical output that can be incorporated with the graphical database developed by the Johns Hopkins APL. Some very specific movement issues could be further developed. In particular, time delays for detection and acceleration could be incorporated. Detection delays are easily added, but the acceleration problem is a significant one, since the computation of an intercept point requires the use of an intercept speed. A multi-step calculation over time seems possible, but somewhat problematic. In any case, it is a problem worth pursuing, since it would add a great deal of realism to the simulation.

The exploratory analysis of the previous chapter provides good first level insights, but is not an exhaustive study of the feasible parameter space. More detailed follow-on analysis should look more closely at response surfaces across a broader range of input parameters to provide more exact findings about how specific changes affect performance. The linear regression models seem to capture a great deal of output variation, but different models, higher order terms, could be investigated. An advantage of the fractional factorial design, is that the experiment need not be re-engineered to allow such deeper analyses. It may well be that follow-on thesis work should be done *using*—rather than developing—this model at the classified level, so more specific questions can be answered.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. JAVA CODE FOR THREAT INTERCEPT POINT

```
/** @param pb the PBMM that controls the PB to be used for the intercept
 * @param threat the threat that needs to be intercepted
 * @ return the Coordinate of the intercept
 * This method determines whether or not a threat can be intercepted by
 * a given PBMM. If so, it passes the intercept point and schedules the end of the
 * intercept at the intercept time calculated within. If not, it returns null
 */
public Coordinate getIntercept(PBMM pb, TypedBasicMover threat){

    tX = threat.getVelocity().getXCoord();
    tY = threat.getVelocity().getYCoord();
    tXPos = threat.getCurrentLocation().getXCoord();
    tYPos = threat.getCurrentLocation().getYCoord();
    pbXPos = pb.getLocation().getXCoord();
    pbYPos = pb.getLocation().getYCoord();
    dX = tXPos - pbXPos;
    dY = tYPos - pbYPos;
    k = dY * tX - dX*tY;
    //define constants of quadratic formula to solve for y component of
    //O/S intercept vector based upon intercept speed.
    a = Math.pow(dY,2.0) + Math.pow(dX,2.0);
    b = 2.0 * dX * k;
    c = Math.pow(k,2.0) - Math.pow(pb.getInterceptSpeed(),2.0)*Math.pow(dY,2.0);
    w = Math.pow(b,2.0) - 4.0*a*c;
    if (w < 0.0){
        firePropertyChange("imaginaryRoot", imaginaryRoot, ++imaginaryRoot);
        return null;
    }
    else{
        pbY1 = (-b +Math.sqrt(w))/(2.0 * a);
        pbY2 = (-b - Math.sqrt(w))/(2.0 * a);
        time1 = dY/(pbY1 -tY);
        time2 = dY/(pbY2 - tY);
    }
}
```

```

if ( (time1 >=0.0) && (time2 >=0.0)){
    interceptTime = Math.min(time1,time2);
    intX = tXPos + interceptTime * tX;
    intY = tYPos + interceptTime * tY;
    Coordinate intPt = new Coordinate(intX,intY);
    if (intX >= 0.0 && intX <= RiverData.RIVER_SIZE_X && intY >=0.0 && intY <=
RiverData.RIVER_SIZE_Y){
        return intPt;
    }
    else{
        firePropertyChange("outsideRiver", outsideRiver, ++outsideRiver);
        return null;
    }
}
else if (time1 >= 0.0){
    interceptTime = time1;
    intX = tXPos + interceptTime * tX;
    intY = tYPos + interceptTime * tY;
    Coordinate intPt = new Coordinate(intX,intY);
    if (intX >= 0.0 && intX <= RiverData.RIVER_SIZE_X && intY >=0.0 && intY <=
RiverData.RIVER_SIZE_Y){
        return intPt;
    }
    else{
        firePropertyChange("outsideRiver", outsideRiver, ++outsideRiver);
        return null;
    }
}
else if (time2 >= 0.0){
    interceptTime = time2;
    intX = tXPos + interceptTime * tX;
    intY = tYPos + interceptTime * tY;
    Coordinate intPt = new Coordinate(intX,intY);
    if (intX >= 0.0 && intX <= RiverData.RIVER_SIZE_X && intY >=0.0 && intY <=
RiverData.RIVER_SIZE_Y){
        return intPt;
    }
}

```

```
        else{
            firePropertyChange("outsideRiver",
outsideRiver, ++outsideRiver);
            return null;
        }
    }
    else {
        firePropertyChange("interceptPassed",
interceptPassed, ++interceptPassed);
        return null;
    }
}
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DESIGN POINT DATA

		breach- count	leakers- count	numberAv- ailablePBs- mean	numberInt- erceptQ- mean	numberInt- ercepted- count	rangeAtInt- ercept- mean	terror- count
dp1.1	mean	2.93988	13.4509	1.830053	0.004025	30.88577	957.3882	0.152305
	sd	1.768872	3.781042	0.051879	0.046242	5.095964	86.19461	0.359676
	var	3.128908	14.29628	0.002691	0.002138	25.96885	7429.511	0.129367
dp1.2	mean	3.661323	14.48297	1.838598	0.001172	30.18437	954.2404	0.861723
	sd	1.632848	3.811472	0.050074	0.002704	5.220641	81.17603	0.345536
	var	2.666192	14.52732	0.002507	7.31E-06	27.2551	6589.548	0.119395
dp1.3	mean	3.881764	14.41683	1.835858	0.00449	31.08617	954.6288	0.991984
	sd	1.68028	3.702083	0.049546	0.049565	5.177821	83.28889	0.533925
	var	2.823341	13.70542	0.002455	0.002457	26.80983	6937.039	0.285076
dp1.4	mean	3.336673	14.12024	1.833563	0.00309	31.42285	960.5784	0.637275
	sd	1.758193	3.925712	0.050049	0.033851	5.381767	81.91754	0.61335
	var	3.091243	15.41122	0.002505	0.001146	28.96341	6710.484	0.376198
dp1.5	mean	4.643287	15.47295	1.837511	0.009546	30.26453	956.5379	1.843687
	sd	1.658921	3.904067	0.048999	0.08931	5.189014	90.34237	0.390159
	var	2.752018	15.24174	0.002401	0.007976	26.92587	8161.745	0.152224
dp1.6	mean	4.521042	15.26052	1.838607	0.002538	30.30661	953.5084	1.851703
	sd	1.648005	3.957133	0.048999	0.020218	5.232753	85.98333	0.372299
	var	2.715922	15.6589	0.002401	0.000409	27.3817	7393.134	0.138607
dp1.7	mean	3.897796	14.52705	1.838172	0.017015	30.94188	953.5896	1.114228
	sd	1.684913	3.894797	0.047951	0.120719	5.283397	83.8448	0.430953
	var	2.838931	15.16945	0.002299	0.014573	27.91429	7029.95	0.185721
dp1.8	mean	3.857715	14.58918	1.836917	0.018639	30.998	956.0409	1.114228
	sd	1.711582	4.034684	0.049164	0.144296	5.309497	82.83713	0.466743
	var	2.929514	16.27868	0.002417	0.020821	28.19076	6861.991	0.217849
	$\bar{Y}_{1.}$	3.842435	14.54008	1.83616	0.007564	30.76127	955.8141	0.598823
	$S^2_{1.}$	2.868259	15.03612	0.00246	0.006191	27.42623	7139.175	0.07346
	Var(means)	0.315703	0.396126	8.93E-06	4.63E-05	0.204812	5.699677	0.081763
	V_1	3.183961	15.43225	0.002469	0.006237	27.63104	7144.875	0.155223

Table 16. Design Point One Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInI nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp2.1	mean	1.521042	7.61523	3.90508	0	31.46293	992.3243	0.01002
	sd	1.23416	2.816376	0.030197	0	5.452251	77.09115	0.099697
	var	1.523151	7.931976	0.000912	0	29.72704	5943.046	0.00994
dp2.2	mean	2.216433	8.490982	3.906153	0	30.88778	978.7149	0.689379
	sd	1.352244	2.925003	0.03176	0	5.472038	76.71705	0.463212
	var	1.828565	8.555641	0.001009	0	29.9432	5885.505	0.214566
dp2.3	mean	1.893788	7.871743	3.899301	2.73E-07	32.25852	982.6089	0.166333
	sd	1.337872	2.807689	0.033242	6.11E-06	5.309999	79.63415	0.378101
	var	1.789901	7.883116	0.001105	3.73E-11	28.19609	6341.598	0.142961
dp2.4	mean	1.523046	7.777555	3.900288	0	32.25852	995.4897	0.038076
	sd	1.217744	2.763906	0.041766	0	5.396647	79.86006	0.201782
	var	1.482902	7.639174	0.001744	0	29.1238	6377.629	0.040716
dp2.5	mean	3.288577	9.52505	3.904803	0	30.89579	978.1589	1.695391
	sd	1.358038	2.864559	0.031397	0	5.397936	83.17954	0.510334
	var	1.844267	8.205697	0.000986	0	29.13771	6918.836	0.260441
dp2.6	mean	3.276553	9.418838	3.90402	0	30.84369	981.2241	1.723447
	sd	1.367902	2.810431	0.036631	0	5.457725	82.5569	0.478106
	var	1.871156	7.89852	0.001342	0	29.78676	6815.641	0.228586
dp2.7	mean	2.276553	8.496994	3.904062	0	31.6012	985.4207	0.777555
	sd	1.24009	2.707548	0.032566	0	5.551446	79.92628	0.430533
	var	1.537823	7.330814	0.001061	0	30.81855	6388.211	0.185359
dp2.8	mean	2.282565	8.593186	3.903893	0	31.57515	985.0167	0.781563
	sd	1.274683	2.967839	0.033127	0	5.39186	78.5618	0.423199
	var	1.624816	8.808066	0.001097	0	29.07215	6171.957	0.179097
	Ybar₂	2.28482	8.473697	3.90345	3.42E-08	31.47295	984.8698	0.411323
	S²₂	1.687822	8.031625	0.001157	4.66E-12	29.47566	6355.303	0.060474
	Var(means	0.473197	0.513954	5.7E-06	9.34E-15	0.333248	38.57139	0.127735
	V₂	2.161019	8.54558	0.001163	4.67E-12	29.80891	6393.874	0.188209

Table 17. Design Point Two Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInl nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp3.1	mean	2.871743	14.14028	1.8374	0.001226	30.8998	967.6103	0.160321
	sd	1.720297	3.821883	0.049928	0.003014	5.152492	86.03063	0.367271
	var	2.959421	14.60679	0.002493	9.08E-06	26.54817	7401.269	0.134888
dp3.2	mean	3.559118	14.48697	1.843198	0.012113	30.31663	961.0372	0.815631
	sd	1.698692	3.889195	0.048941	0.140274	5.214663	79.47004	0.388174
	var	2.885554	15.12583	0.002395	0.019677	27.19271	6315.487	0.150679
dp3.3	mean	3.805611	14.89178	1.838992	0.002014	31.12425	957.7534	0.97996
	sd	1.687507	3.796573	0.046642	0.010686	5.275216	83.19879	0.623824
	var	2.847679	14.41397	0.002175	0.000114	27.8279	6922.039	0.389156
dp3.4	mean	3.396794	14.56513	1.837979	0.001562	31.42886	969.8448	0.643287
	sd	1.750178	3.860685	0.048068	0.00323	5.221639	81.58087	0.662393
	var	3.063122	14.90489	0.002311	1.04E-05	27.26551	6655.438	0.438765
dp3.5	mean	4.655311	15.60922	1.840242	0.002259	30.37275	958.7996	1.817635
	sd	1.690223	3.818365	0.05098	0.020226	5.366239	82.76137	0.430757
	var	2.856854	14.57991	0.002599	0.000409	28.79652	6849.445	0.185552
dp3.6	mean	4.657315	15.61323	1.840242	0.002259	30.36874	958.884	1.821643
	sd	1.685278	3.819826	0.05098	0.020226	5.36165	82.86046	0.418273
	var	2.840162	14.59107	0.002599	0.000409	28.74729	6865.856	0.174952
dp3.7	mean	3.895792	14.88577	1.841277	0.001526	31.00601	954.4103	1.082164
	sd	1.697258	3.813548	0.046378	0.004028	5.324978	78.1647	0.477677
	var	2.880685	14.54315	0.002151	1.62E-05	28.35539	6109.721	0.228175
dp3.8	mean	3.895792	14.88577	1.841277	0.001526	31.00601	954.4103	1.082164
	sd	1.697258	3.813548	0.046378	0.004028	5.324978	78.1647	0.477677
	var	2.880685	14.54315	0.002151	1.62E-05	28.35539	6109.721	0.228175
	Ybar₃	3.842184	14.88477	1.840076	0.003061	30.81538	960.3437	0.586172
	S²₃	2.90177	14.6636	0.002359	0.002583	27.88611	6653.622	0.087095
	Var(means	0.364707	0.267172	3.63E-06	1.35E-05	0.170818	32.16972	0.075256
	V₃	3.266478	14.93077	0.002363	0.002596	28.05693	6685.792	0.162351

Table 18. Design Point Three Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInl nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp4.1	mean	1.488978	6.164329	3.897723	0	31.30862	987.7685	0.004008
	sd	1.181084	2.408371	0.033435	0	5.408693	77.20181	0.063245
	var	1.394959	5.800251	0.001118	0	29.25396	5960.119	0.004
dp4.2	mean	2.344689	6.881764	3.896577	0	30.5992	976.4362	0.709419
	sd	1.320759	2.546761	0.035509	0	5.445207	81.30257	0.454486
	var	1.744404	6.485992	0.001261	0	29.65028	6610.108	0.206558
dp4.3	mean	1.875752	6.370741	3.894981	0	32.1523	979.0659	0.152305
	sd	1.261331	2.512784	0.030843	0	5.400882	80.3636	0.365217
	var	1.590957	6.314082	0.000951	0	29.16953	6458.308	0.133383
dp4.4	mean	1.511022	6.206413	3.892906	2.82E-06	32.37275	991.675	0.018036
	sd	1.221206	2.534002	0.035348	6.30E-05	5.400556	76.33099	0.147521
	var	1.491344	6.421164	0.001249	3.97E-09	29.166	5826.42	0.021762
dp4.5	mean	3.346693	7.783567	3.899175	0	30.62725	974.9802	1.669339
	sd	1.308776	2.543053	0.030912	0	5.381933	85.87928	0.51179
	var	1.712896	6.467119	0.000956	0	28.9652	7375.251	0.261929
dp4.6	mean	3.322645	7.951904	3.899218	0	30.76152	978.9052	1.693387
	sd	1.285581	2.651379	0.030281	0	5.543298	80.51856	0.526579
	var	1.652719	7.029811	0.000917	0	30.72816	6483.238	0.277285
dp4.7	mean	2.374749	6.977956	3.895178	0	31.62725	982.1045	0.795591
	sd	1.23935	2.671716	0.032747	0	5.390135	80.47742	0.403674
	var	1.535988	7.138067	0.001072	0	29.05355	6476.616	0.162952
dp4.8	mean	2.382766	7.128257	3.895927	0	31.55311	985.4959	0.821643
	sd	1.26419	2.64719	0.034487	0	5.445853	76.02263	0.388402
	var	1.598176	7.007614	0.001189	0	29.65731	5779.44	0.150856
	Y_4	2.330912	6.933116	3.896461	3.52E-07	31.37525	982.0539	0.411072
	S^2_4	1.59018	6.583012	0.001089	4.96E-10	29.4555	6371.188	0.057825
	Var(means	0.515484	0.46276	4.76E-06	9.94E-13	0.462401	33.93009	0.128273
	V_4	2.105665	7.045772	0.001094	4.97E-10	29.9179	6405.118	0.186097

Table 19. Design Point Four Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInI nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp5.1	mean	1.406814	8.436874	1.89181	3.53E-04	31.40681	988.049	0.076152
	sd	1.191011	2.976063	0.032411	0.001124	5.339934	79.05218	0.265508
	var	1.418508	8.856951	0.00105	1.26E-06	28.51489	6249.247	0.070494
dp5.2	mean	2.172345	9.164329	1.892792	3.08E-04	30.83768	980.3963	0.757515
	sd	1.258847	3.169421	0.034096	0.001055	5.408207	86.98555	0.429016
	var	1.584695	10.04523	0.001163	1.11E-06	29.2487	7566.486	0.184055
dp5.3	mean	2.170341	8.997996	1.886903	0.006115	31.71543	981.4791	0.659319
	sd	1.237403	2.935381	0.036688	0.072923	5.371274	80.87712	0.552621
	var	1.531167	8.616462	0.001346	0.005318	28.85058	6541.108	0.30539
dp5.4	mean	1.785571	8.567134	1.888897	0.019514	31.96393	994.7002	0.432866
	sd	1.286364	3.087252	0.033733	0.134813	5.347625	78.28273	0.567707
	var	1.654731	9.531127	0.001138	0.018174	28.59709	6128.185	0.322291
dp5.5	mean	3.128257	9.903808	1.894741	3.41E-04	30.73547	976.6556	1.729459
	sd	1.217173	3.119888	0.034573	0.001113	5.331027	82.65811	0.471
	var	1.481509	9.7337	0.001195	1.24E-06	28.41984	6832.363	0.221841
dp5.6	mean	3.082164	9.957916	1.893316	3.94E-04	30.68537	980.6809	1.699399
	sd	1.2318	2.993338	0.032584	0.001559	5.296737	79.11845	0.480357
	var	1.517332	8.960073	0.001062	2.43E-06	28.05543	6259.73	0.230743
dp5.7	mean	2.426854	9.222445	1.890801	3.96E-04	31.52305	986.2909	0.953908
	sd	1.170316	2.956956	0.0325	0.001295	5.44717	87.42914	0.374755
	var	1.369639	8.743592	0.001056	1.68E-06	29.67166	7643.854	0.140442
dp5.8	mean	2.372745	9.308617	1.891251	2.80E-04	31.46894	985.898	0.95992
	sd	1.214594	3.096336	0.031641	8.86E-04	5.385588	82.27686	0.378112
	var	1.47524	9.587295	0.001001	7.84E-07	29.00456	6769.482	0.142969
	Y_5	2.318136	9.19489	1.891314	0.003463	31.29208	984.2688	0.506388
	S^2_5	1.504103	9.259304	0.001126	0.002938	28.79534	6748.807	0.074434
	Var(means	0.344581	0.300976	6.25E-06	4.61E-05	0.230398	31.92297	0.08733
	V_5	1.848684	9.56028	0.001133	0.002984	29.02574	6780.73	0.161764

Table 20. Design Point Five Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInl nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp6.1	mean	0.681363	2.274549	3.931576	0	31.4509	1020.674	0
	sd	0.866995	1.563506	0.023759	0	5.455469	78.84387	0
	var	0.75168	2.444552	0.000564	0	29.76214	6216.356	0
dp6.2	mean	1.368737	2.743487	3.932368	0	31.03006	999.7258	0.460922
	sd	0.889646	1.637393	0.024151	0	5.49453	85.07972	0.498971
	var	0.79147	2.681057	0.000583	0	30.18986	7238.559	0.248972
dp6.3	mean	0.825651	2.326653	3.928298	0	32.56112	1014.396	0.018036
	sd	0.877922	1.535232	0.023316	0	5.355042	79.51323	0.133215
	var	0.770746	2.356939	0.000544	0	28.67648	6322.354	0.017746
dp6.4	mean	0.699399	2.330661	3.928148	0	32.48096	1027.27	0.012024
	sd	0.865341	1.456491	0.027545	0	5.43242	79.62561	0.109102
	var	0.748815	2.121367	0.000759	0	29.51118	6340.237	0.011903
dp6.5	mean	2.286573	3.601202	3.930157	0	31.08016	996.812	1.342685
	sd	0.918376	1.589144	0.029879	0	5.375236	78.69261	0.646897
	var	0.843414	2.52538	0.000893	0	28.89316	6192.526	0.418476
dp6.6	mean	2.418838	3.677355	3.932861	0	31.0521	1000.355	1.440882
	sd	0.975053	1.644667	0.022702	0	5.445356	82.96707	0.609506
	var	0.950729	2.704928	0.000515	0	29.6519	6883.534	0.371498
dp6.7	mean	1.450902	2.777555	3.930874	0	31.87976	1012.529	0.579158
	sd	0.88216	1.5807	0.023089	0	5.392019	84.39548	0.49419
	var	0.778207	2.498612	0.000533	0	29.07387	7122.598	0.244223
dp6.8	mean	1.521042	2.875752	3.930306	0	31.86573	1015.996	0.591182
	sd	0.918932	1.640536	0.025377	0	5.462174	81.73058	0.492109
	var	0.844436	2.691359	0.000644	0	29.83535	6679.888	0.242171
	Y_6	1.406563	2.825902	3.930574	0	31.6751	1010.97	0.306613
	S^2_6	0.809937	2.503024	0.000629	0	29.44924	6624.507	0.071935
	Var(means	0.454154	0.305169	2.98E-06	0	0.388728	119.8325	0.08592
	V_6	1.264091	2.808193	0.000632	0	29.83797	6744.339	0.157854

Table 21. Design Point Six Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInl nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp7.1	mean	1.645291	7.759519	1.89009	4.28E-04	31.42285	1004.647	0.072144
	sd	1.240398	2.957918	0.032279	0.001552	5.346203	78.52796	0.258986
	var	1.538587	8.749282	0.001042	2.41E-06	28.58189	6166.641	0.067074
dp7.2	mean	2.400802	8.242485	1.892843	0.006418	30.82164	997.5002	0.719439
	sd	1.313996	2.873385	0.032574	0.077189	5.409186	82.98657	0.449724
	var	1.726586	8.256344	0.001061	0.005958	29.25929	6886.771	0.202252
dp7.3	mean	2.348697	8.106212	1.888295	4.01E-04	31.82966	997.2305	0.651303
	sd	1.354249	2.826073	0.033307	0.001325	5.372569	82.49487	0.626276
	var	1.833989	7.986688	0.001109	1.76E-06	28.8645	6805.403	0.392222
dp7.4	mean	2.072144	8.012024	1.887579	5.33E-04	31.97796	1006.89	0.49499
	sd	1.418044	2.997967	0.035824	0.001563	5.53063	80.68806	0.625331
	var	2.010849	8.987807	0.001283	2.44E-06	30.58787	6510.564	0.391039
dp7.5	mean	3.368737	9.136273	1.891603	0.008529	30.80962	995.1107	1.695391
	sd	1.312706	2.842143	0.037835	0.089369	5.294332	75.73676	0.49839
	var	1.723197	8.077778	0.001431	0.007987	28.02995	5736.056	0.248392
dp7.6	mean	3.322645	9.172345	1.890531	3.52E-04	30.82365	996.1598	1.647295
	sd	1.232959	2.800308	0.035308	0.001272	5.347837	79.90489	0.526266
	var	1.520189	7.841724	0.001247	1.62E-06	28.59936	6384.792	0.276956
dp7.7	mean	2.60521	8.262525	1.890279	4.35E-04	31.6513	995.2613	0.895792
	sd	1.333253	2.918783	0.03235	0.001439	5.468296	82.6784	0.500163
	var	1.777563	8.519296	0.001047	2.07E-06	29.90226	6835.718	0.250163
dp7.8	mean	2.643287	8.292585	1.88868	3.82E-04	31.57916	994.0101	0.881764
	sd	1.311446	2.686902	0.033377	0.001321	5.551805	80.11691	0.518772
	var	1.71989	7.219443	0.001114	1.75E-06	30.82254	6418.719	0.269125
	Y_7	2.550852	8.372996	1.889988	0.002185	31.36448	998.3512	0.490606
	S^2_7	1.731356	8.204795	0.001167	0.001745	29.33096	6468.083	0.090787
	Var(means	0.341241	0.262154	3.08E-06	1.1E-05	0.231364	22.60338	0.079502
	V_7	2.072597	8.466949	0.00117	0.001756	29.56232	6490.686	0.170289

Table 22. Design Point Seven Data

		breach- count	leakers- count	numberAv ailablePBs mean	numberInl nterceptQ- mean	numberInt ercepted- count	rangeAtInt ercept- mean	terror- count
dp8.1	mean	0.723447	2.731463	3.938206	0	31.65932	1024.497	0
	sd	0.857094	1.541928	0.01898	0	5.419748	83.26708	0
	var	0.73461	2.377542	0.00036	0	29.37366	6933.407	0
dp8.2	mean	1.382766	3.462926	3.938368	0	30.84369	1008.828	0.651303
	sd	0.981651	1.730928	0.022668	0	5.480122	86.13998	0.477036
	var	0.963638	2.996113	0.000514	0	30.03174	7420.095	0.227564
dp8.3	mean	0.851703	2.947896	3.933422	0	32.52104	1014.804	0.014028
	sd	0.919475	1.705557	0.022237	0	5.410931	78.23498	0.117724
	var	0.845434	2.908926	0.000494	0	29.27817	6120.712	0.013859
dp8.4	mean	0.733467	2.849699	3.936044	0	32.46293	1027.678	0.01002
	sd	0.82573	1.669312	0.021015	0	5.435652	80.46576	0.099697
	var	0.68183	2.786601	0.000442	0	29.54631	6474.739	0.00994
dp8.5	mean	2.438878	4.402806	3.937111	0	30.92184	1002.08	1.613226
	sd	0.9641	1.745928	0.023351	0	5.357491	84.14335	0.530879
	var	0.92949	3.048265	0.000545	0	28.70271	7080.103	0.281833
dp8.6	mean	2.380762	4.358717	3.937835	0	30.87575	1005.98	1.609218
	sd	1.023475	1.723211	0.023832	0	5.480578	85.7869	0.550278
	var	1.047501	2.969457	0.000568	0	30.03674	7359.392	0.302806
dp8.7	mean	1.498998	3.54509	3.936707	0	31.93988	1020.433	0.719439
	sd	0.959783	1.671721	0.022591	0	5.454484	83.30476	0.449724
	var	0.921184	2.79465	0.00051	0	29.7514	6939.683	0.202252
dp8.8	mean	1.496994	3.547094	3.93666	0	31.93988	1021.278	0.719439
	sd	0.953482	1.671665	0.022648	0	5.470659	83.82859	0.449724
	var	0.909127	2.794464	0.000513	0	29.92811	7027.232	0.202252
	Y_8	1.438377	3.480711	3.936794	0	31.64554	1015.697	0.374248
	S^2_8	0.879102	2.834502	0.000493	0	29.58111	6919.42	0.0601
	Var(means	0.4645	0.410495	2.52E-06	0	0.481052	86.05511	0.122942
	V_8	1.343602	3.244997	0.000496	0	30.06216	7005.476	0.183041

Table 23. Design Point Eight Data

APPENDIX C LINEAR REGRESSION

The following overview is taken from Hamilton's *Regression With Graphics: A Second Course in Applied Statistics*, chapter two.[Ref. 26]

One of the simplest ways of relating two variables, X and Y , to each other in a mathematical model is with a *linear model*. The linear model is merely an expansion of the equation for a line, which was introduced to most of us at some point, in junior high school or so, as $Y = mX + b$, with Y the value of Y , X the value of X , m the slope of the line, and b the Y -intercept of the line. In this form, X is the *predictor* of Y , in that X is the cause and Y the effect— $Y=f(X)$. The linear regression model is an expansion because it allows for the same sort of linear expression with terms that can be multi-dimensional. $Y = mX + b$ becomes $Y_i = \beta_0 + \beta_i X_i$, with X still the predictor of Y , β_0 now the Y -intercept, and β_i the slope. The i subscript denotes individual observations.

The linear equation described is nothing other than the line that best fits the data collected. The essential means of determining how well a linear regression model approximates the actual response of Y to changes in X is the comparison of the model's predicted Y values (Y^p_i) to the actual Y values (Y_i) in from the data. The difference between each predicted Y value and its actual value is called a *residual error*. Since a prediction can be either above or below the actual value, the residual errors are squared, so they are all positive, and then all the squared residual errors are added to give an overall measure of the goodness of the model in predicting Y given X , called the *sum of squared residuals* (RSS). More compactly:

$$RSS = \sum e^2_i = \sum (Y_i - Y^p_i)^2$$

The lower the value of RSS, the closer the fit between predictions and data. This leads to the *Ordinary Least Squares* (OLS) criterion for selection of β values, which is simply to select the values of β_i that yield the lowest RSS.

Another key measurement of a linear model's goodness is the *coefficient of determination*, R^2 , that gives the proportion of the variance of Y that is explained by X :

$$R^2 = \text{explained variance/total variance} = s^2_{y\text{-pred}}/s^2_y$$

R^2 ranges in value from 0 to 1, with a value of 0 meaning that the changes in the variable X account for none of the changes in Y, and a value of 1 meaning that changes in the variable X completely explain the changes in Y. As the sample variance of the model's predicted Y values gets closer to that of the actual Y values, the model is more closely replicating the reality it is modeling.

R^2 is the "quick look" value that tells whether or not a linear model fits the data well, but the importance of individual variables within the model can only be ascertained through an examination of their individual p-values, which are generated in any regression software output—including Excel. The p-value reflects a given variable's t-statistic, and gives a measure of how likely it is that a given variable's effect upon output is statistically significant. If a variable's effect is statistically significant, that means that its slope term (β_i) is non zero, indicating it should be included in the linear model describing changes in Y.

The development and analysis of regression models is, in practice, more of an art than a science, but the basic indicators of R^2 and individual p-values go a long way toward determining the goodness of a given result.

APPENDIX D. OUTPUT GRAPHICS

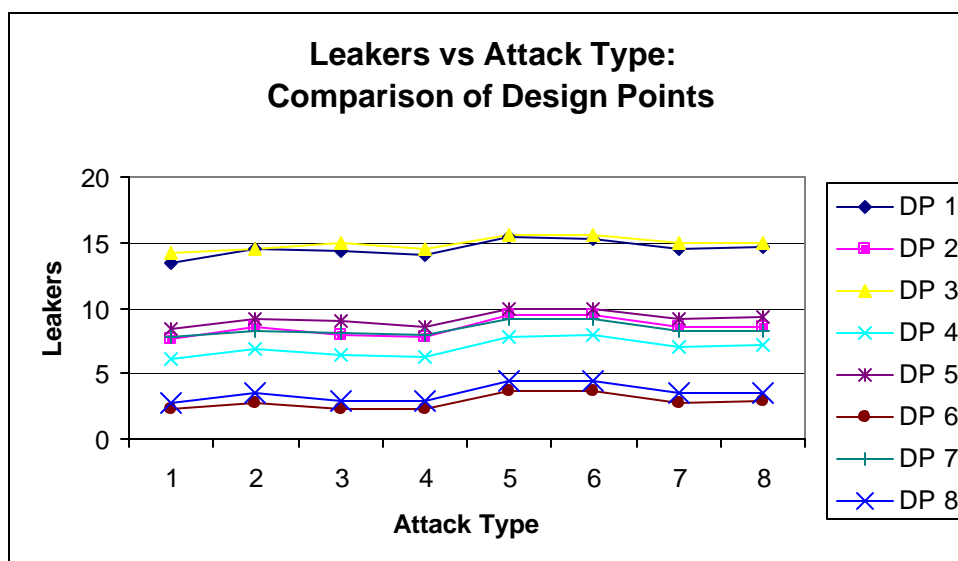


Figure 24. Design Point Six Always Best

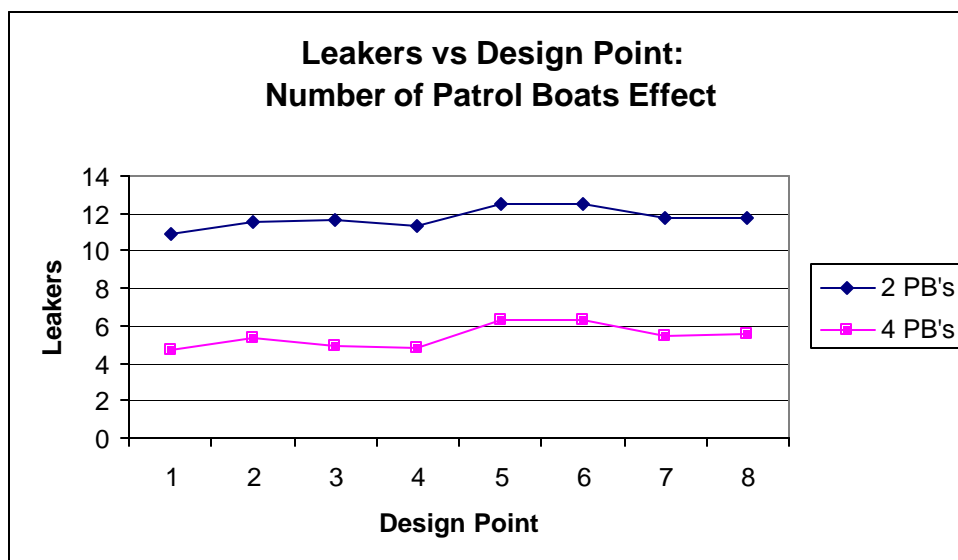


Figure 25. Advantage of Four Patrol Boats Over Two

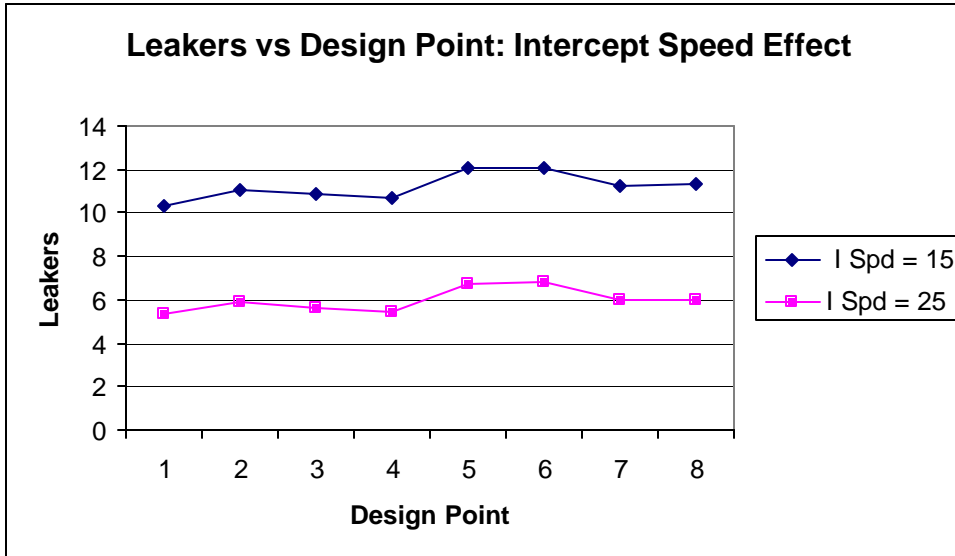


Figure 26. Advantage of Higher Intercept Speed

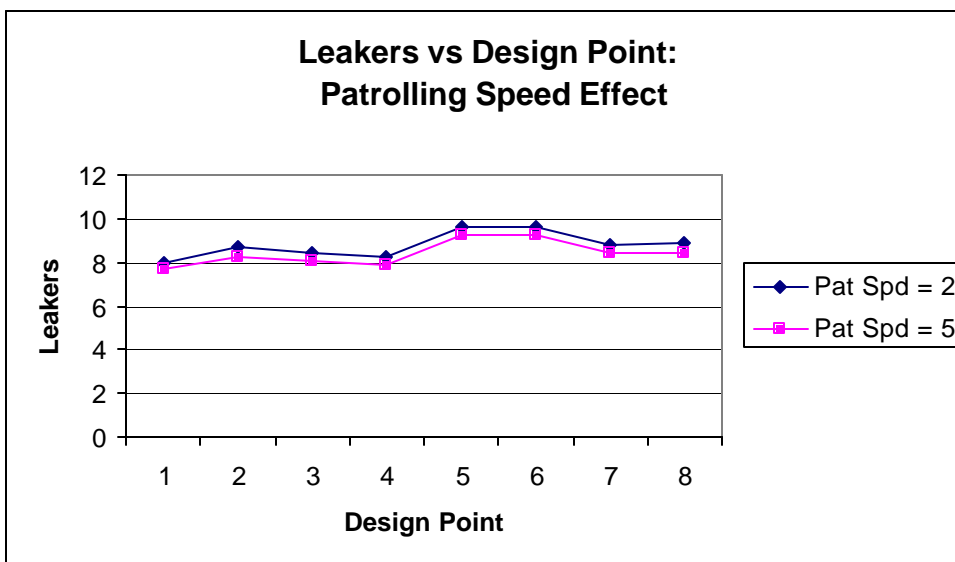


Figure 27. No Clear Difference In Output

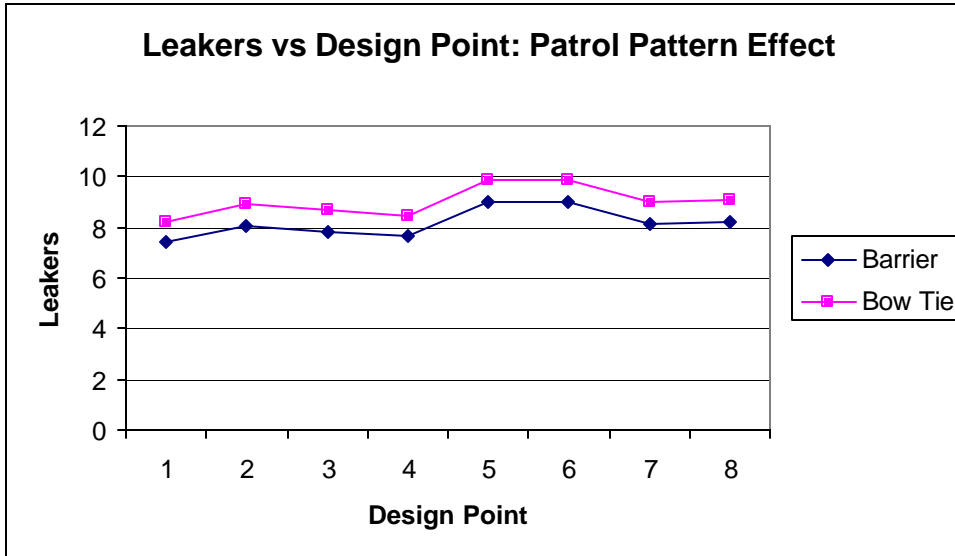


Figure 28. No Clear (Significant) Difference in Output

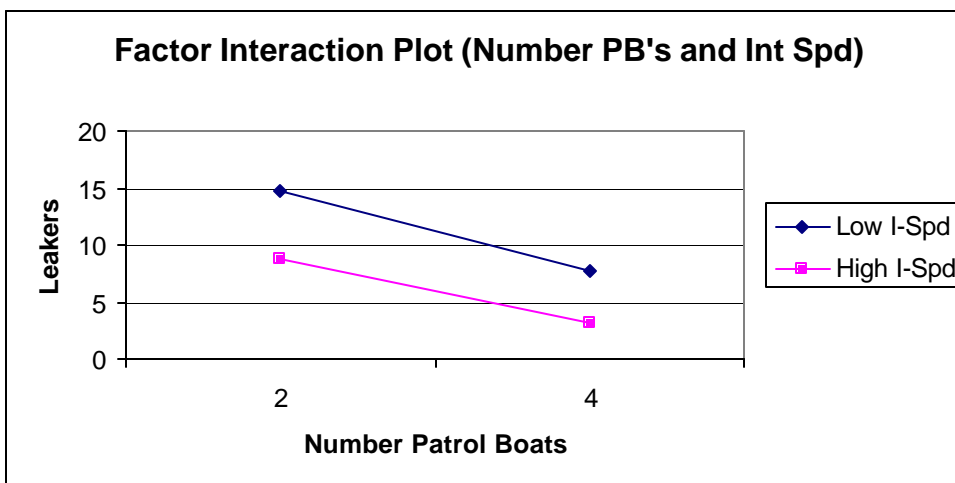


Figure 29. No Interaction Between Significant Factors

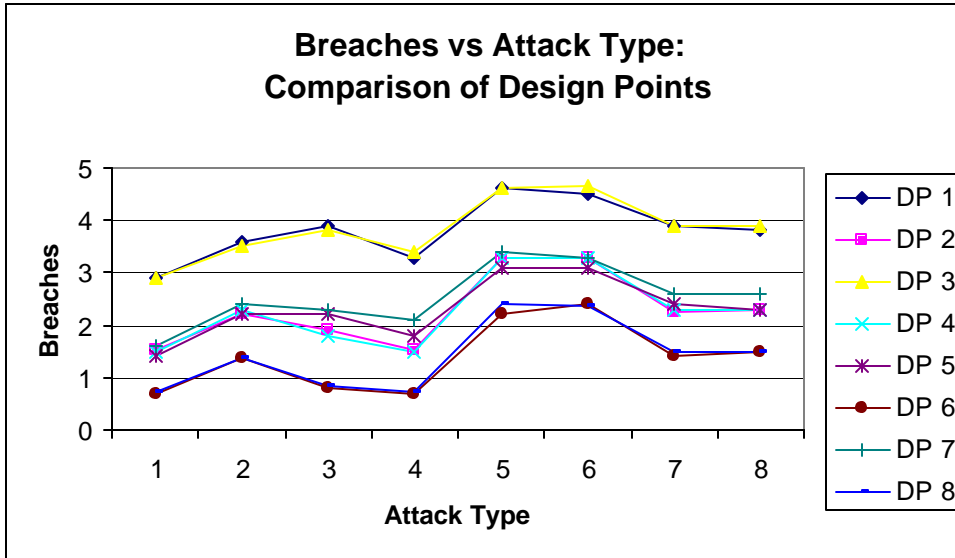


Figure 30. Design Points Six and Eight Are Best

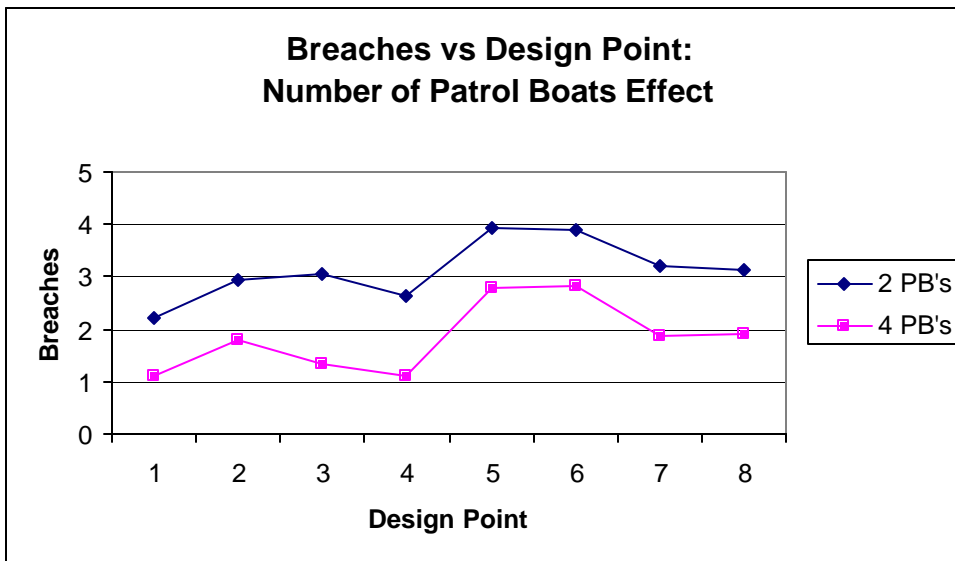


Figure 31. Advantage of Four Patrol Boats Over Two

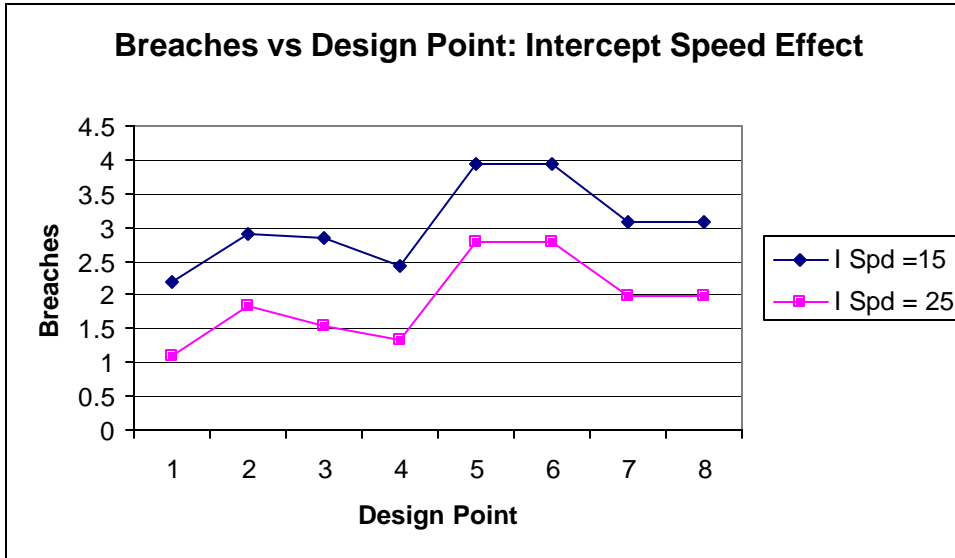


Figure 32. Advantage of Higher Intercept Speed

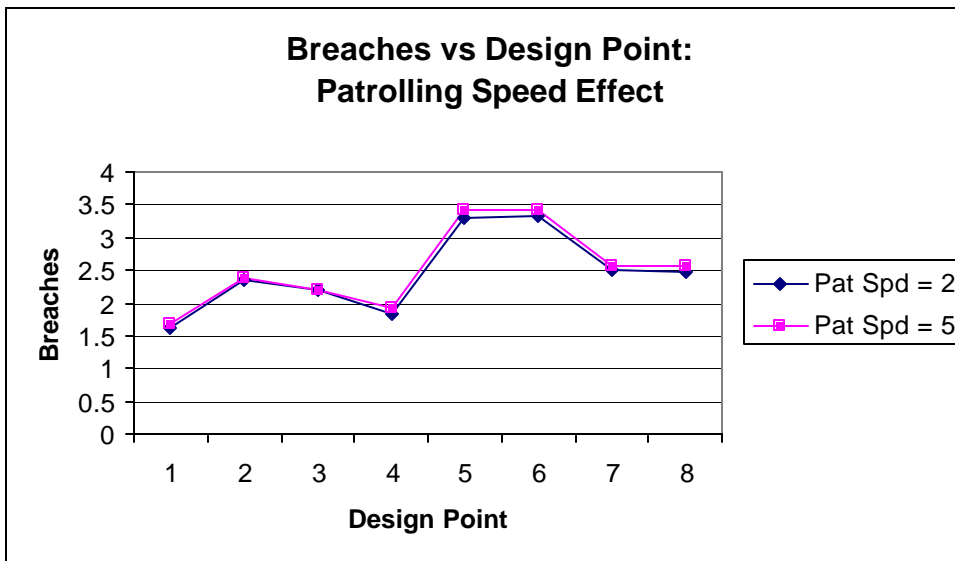


Figure 33. No Clear Difference in Output

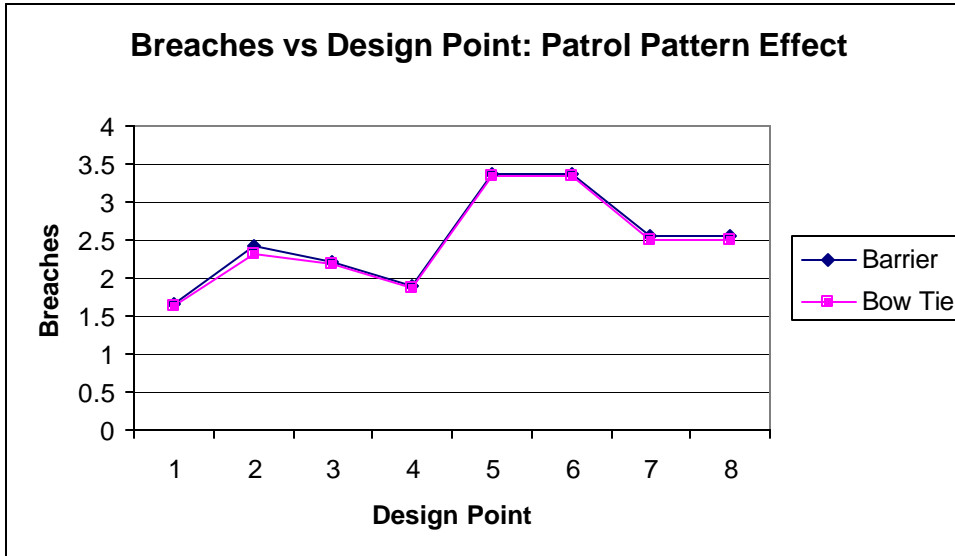


Figure 34. No Clear Difference in Output

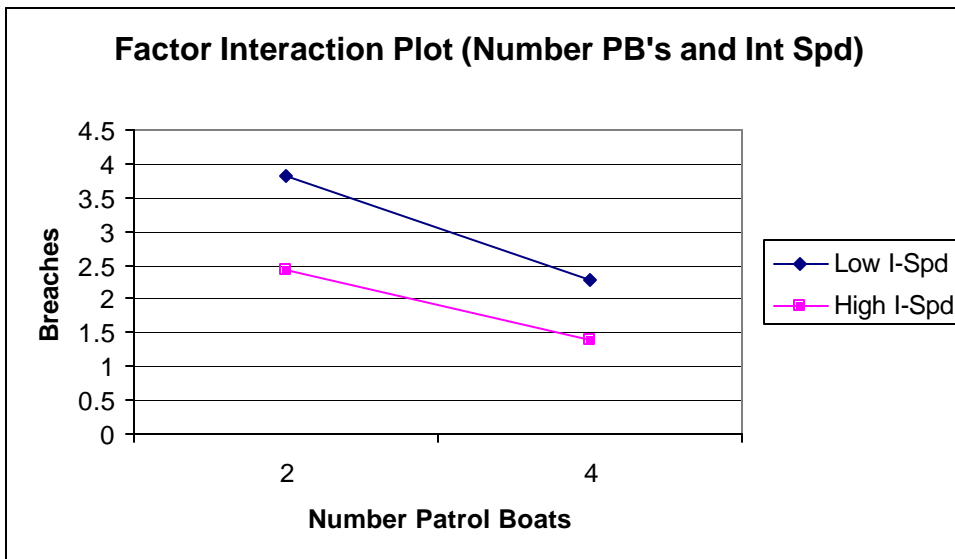


Figure 35. No Interaction Between Significant Factors

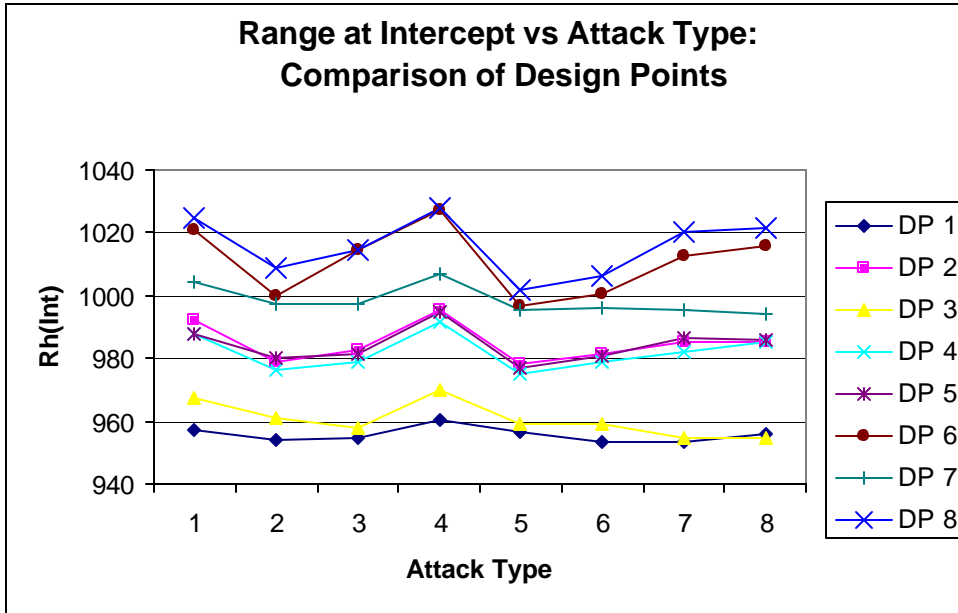


Figure 36. Design Points Six and Eight Are Best

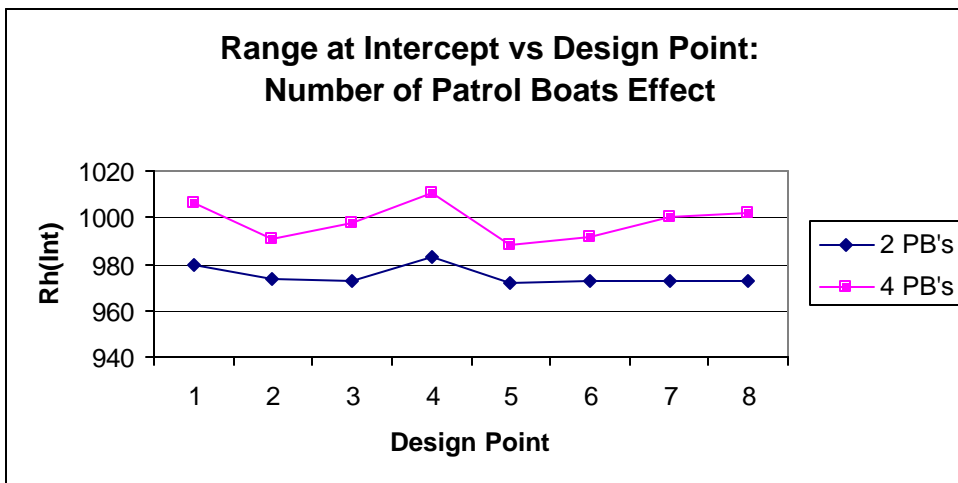


Figure 37. Advantage of Four Patrol Boats Over Two

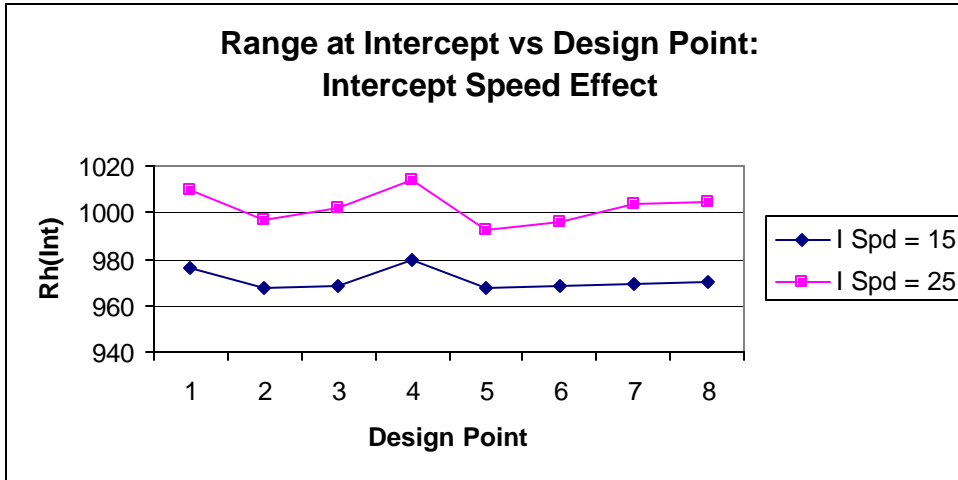


Figure 38. Advantage of Higher Intercept Speed

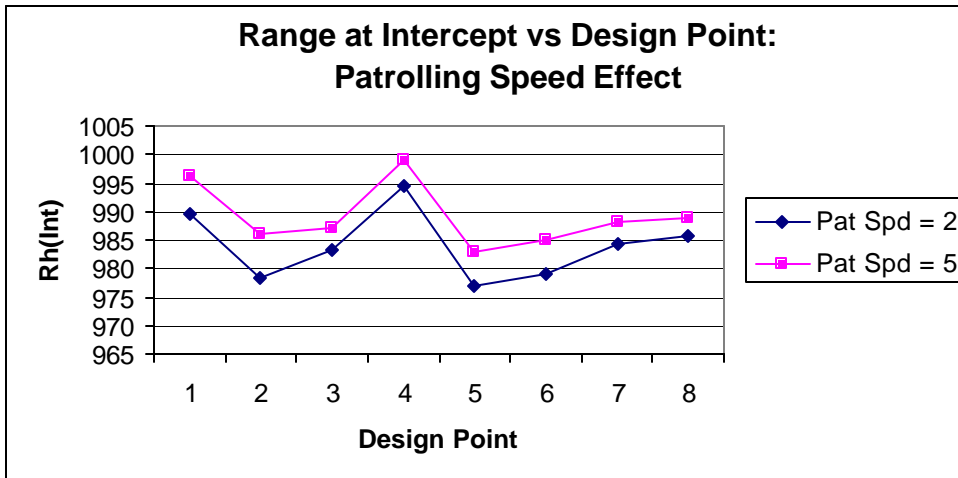


Figure 39. No Clear (Significant) Difference in Output

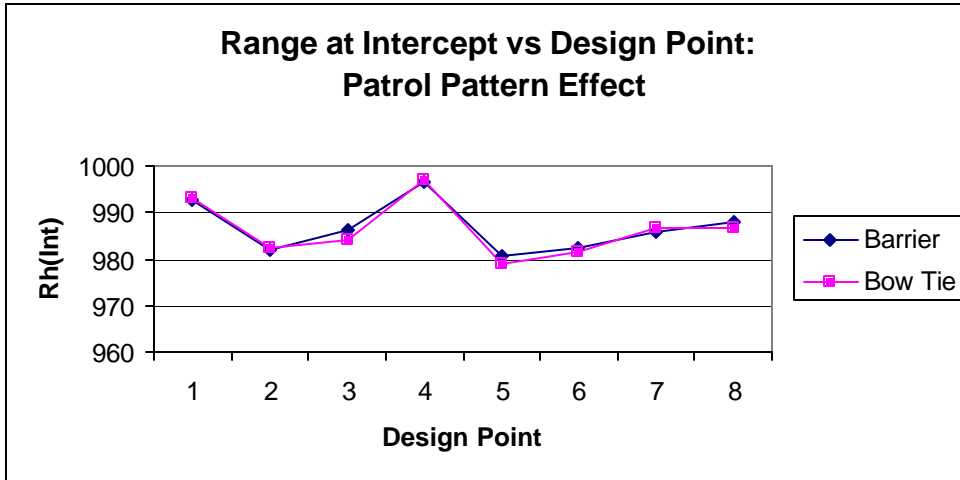


Figure 40. No Clear Difference in Output

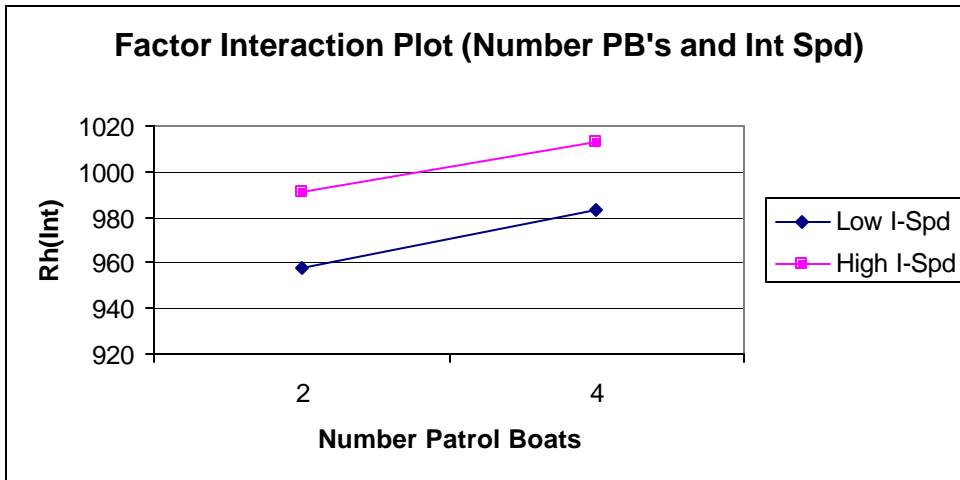


Figure 41. No Interaction Between Significant Factors

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. H. Shelton, Joint Vision 2020, 2000
2. P. Odeen, Transforming Defense: National Security in the 21st Century, Washington, DC 1997
3. M. Van Crevald, Future War, in Strategy and Force Planning, Newport 2000
4. A. Carter, J. Deutch, P. Zelikow, Catastrophic Terrorism: Tackling the New Danger, in Foreign Affairs, November/December 1998
5. H. Gehman , Lost Patrol: The Attack on the *Cole*, in Proceedings, Annapolis Apr 2001
6. D. Weeks, Danger Beyond the Pier, in Proceedings, Annapolis Apr 2001
7. Chief of Naval Operations Message via email correspondence from Commander Submarine Group Nine Security Officer, October 2001. Comm (360) 396-6949
8. Johns Hopkins University Applied Physics Laboratory, Task Assignment Revision and Cost Estimate for the SSBN Security Program, Contract N00024-98-D-8124, Baltimore 1999
9. CSR 16 Message to his Commanding Officers via email correspondence from Executive Officer USS Alabama (Blue), October 2001. Comm (360) 396-8345
10. A. M. Law, W. D. Kelton, Simulation Modeling and Analysis, Boston 2000
11. email correspondence from Commander Submarine Group Nine Security Officer, June 2001. Comm (360) 396-6949.
12. Java available at <http://java.sun.com>
13. Simkit available at <http://Diana.org.nps.navy.mil/Simkit>
14. K. Stork, Sensors in Object Oriented Discrete Event Simulation, Monterey 1996.
15. D. Barnes, Object-Oriented Programming with Java: An Introduction, New Jersey 2000

16. A. Buss, Discrete Event Programming with Simkit, in Simulation News Europe, August 2001.
17. L. Schruben, Simulation Modeling with Event Graphs, in Communications of the ACM 26, 1983
18. A. Buss, Basic Event Graph Modeling, Monterey, CA, 2001
19. A. Buss, Component-Based Simulation Modeling, in Proceedings of the 2000 Winter Simulation Conference, 2000
20. A. Buss, Simple Movement and Detection in Discrete-Event Simulation, in Class Handout for OA 3302, Winter 2001
21. P. Bratley, B. Fox, L. Schrage, A Guide to Simulation, New York 1987
22. DoD Directive 5000.59, DoD Modeling and Simulation (M&S) Management, 1997
23. S. Ross, Introduction to Probability Models, San Diego 1997
24. G. Box, W. Hunter, J. Hunter, Statistics for Experimenters, New York 1978
25. S. Sanchez, P. Sanchez, J. Ramberg, A Framework for Robust System and Process Design Through Simulation, in Concurrent Design of Products, Manufacturing Processes and Systems, New York 1998
26. L. Hamilton, Regression with Graphics: A Second Course in Applied Statistics, Belmont, CA 1992

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. CDR David Mackovjak
Submarine Group Nine
Bangor, Washington
4. Erik W. Johnson
Johns Hopkins University Applied Physics Laboratory
Submarine Technology Department
Operational Assessments Group
Baltimore, Maryland
5. CDR (sel) James C. Childs
USS Alabama (BLUE)
Bangor, Washington
6. Professor Arnold Buss
Department of Operations Research
Naval Postgraduate School
Monterey, California
7. Commander Matthew Boensel
Department of Operations Research
Naval Postgraduate School
Monterey, California
8. Kevin Downer
USCG Research Center
Groton, CT